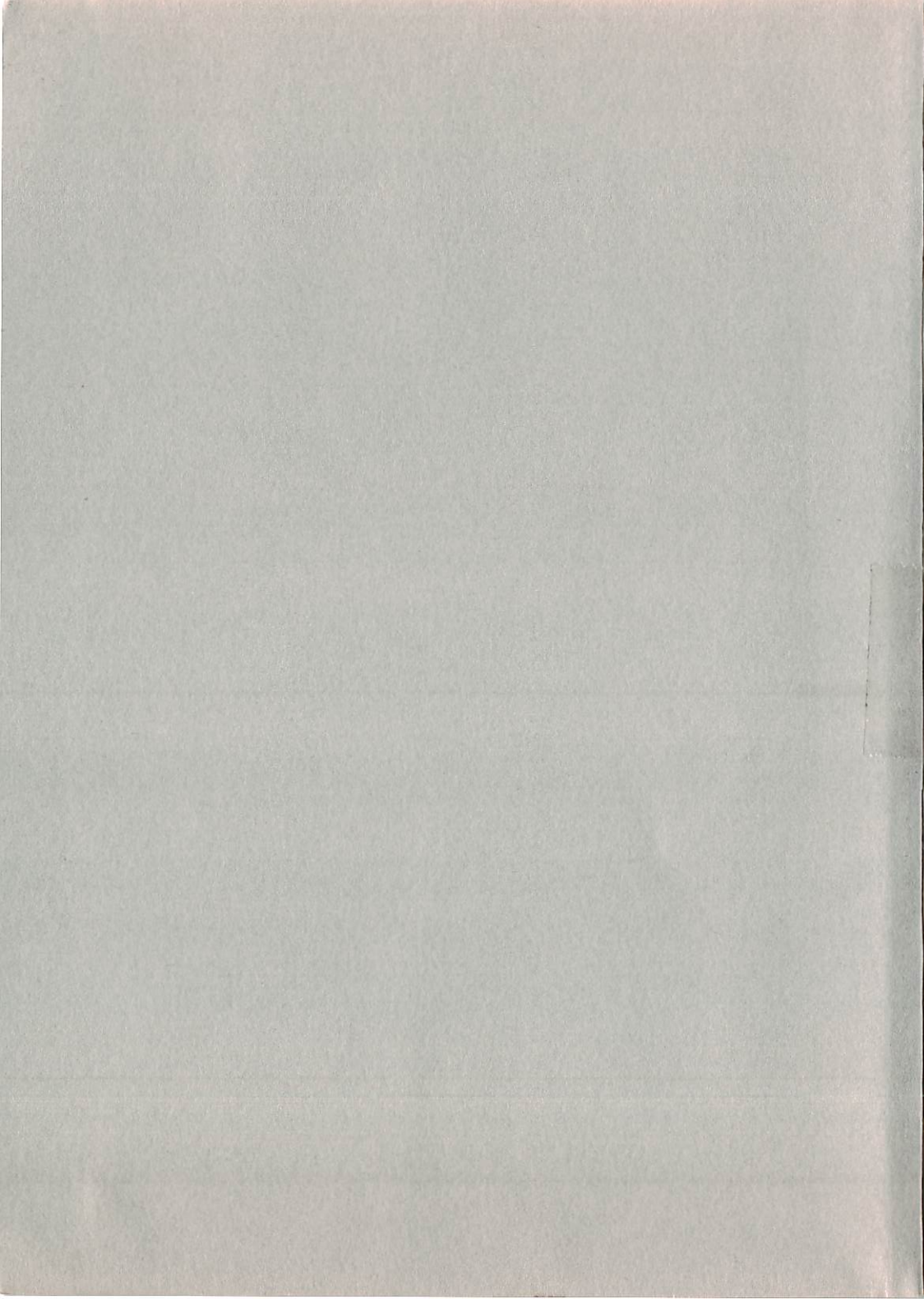
MSX-DOS アセンブラ プログラミング

蔭山哲也 著

MSX-DOS アセンブラ プログラミング
蔭山哲也 著

ASCII
ASCII BOOKS

アスキー出版局

MSX-DOS アセンブラ プログラミング 蔭山哲也 著

ASCII
ASCII BOOKS

MSX-DOS
アセンブラ プログラミング

蔭山哲也 著

アスキー出版局

MSX-DOS アセンブラ プログラミング

蔭山哲也　著

アスキー出版局

はじめに

MSXは，1983年6月に“ホーム・パーソナルコンピュータ・システム”の統一仕様として発表されました．優秀なゲームが数多く発表されたこともあって，当初はゲーム機として評価されることが多かったようです．

ところが，最近MSXにとって大きな2つの動きがありました．その1つは，ハードウェアの低価格化です．この結果，MSXの上位互換機であるMSX2も身近なものとなり，同時にディスク・ドライブも普及してきました．そしてもう1つの動きは，プログラム開発環境の整備，具体的には“MSX-DOS TOOLS”というソフトウェア・パッケージの発売です．これらの動きは，MSXが単なるゲーム機ではないことを証明したものだといってもよいでしょう．

ところで，そのMSXをパーソナルコンピュータとして使う醍醐味は，なんといっても自分でプログラムを作り，それを使うところにあります．しかし，実用に耐えうるプログラムを作るためにはマシン語を使う必要があります．いままでにも，BASIC上でマシン語プログラムを作ることはできましたが，MSX-DOSというプログラム開発環境が整った現在，これを利用しない手はありません．

本書では，マクロアセンブラを始めとするMSX-DOS上のソフトウェア・ツールの使い方や，MSX-DOSでのプログラム開発に欠かせないシステムコールの利用法などについて解説しています．本書の主眼は，マシン語のニーモニックや，プログラミングの細かい技法ではなく，プログラムを作る際に注意すべきルールや，実践的なプログラミングにあります．

本書によって，1人でも多くのMSXユーザーにプログラミングの楽しさをわかってもらえるようにと願っています．

最後に，筆者の限界を超えた遅筆で多大のご迷惑をかけた第一書籍編集部の佐藤英一氏および担当の竹内充彦氏，また数多くの図版を作成してくれた中田秀基氏に深く感謝します．

1988年5月　蔭山哲也

本書を読む前に

本書は，Z80CPUのマシン語とMSX-DOSについての基礎知識を前提に書かれています．これらの基礎知識のない人は，次にあげるような入門書を1冊読んでおくとよいでしょう．

●マシン語について

・MSXマシン語入門講座

・パワーアップマシン語入門

・MSXポケットバンク マシン語入門／同PART2

● MSX-DOSについて

・MSX-DOS入門

（以上いずれもアスキー出版局）

本書で使用するハードウェアおよびソフトウェア

本書では以下のハードウェア，ソフトウェアを使用します．

●ハードウェア

・MSX本体（RAM64Kバイト以上）

・ディスク・ドライブ

・ディスプレイ

●ソフトウェア

・MSX-DOS TOOLS

（株）アスキー　価格14,800円

・MSX-S BUG（6章でのみ使用）

（株）アスキー　価格19,800円

目　次

3章 アセンブリ言語での約束事 57
—書式，擬似命令，マクロ機能—

4章 プログラム開発の効率化 83
—プログラムをモジュール別に開発する—

5章 システムコールの活用 111
—ファイルの入出力を行う—

6章 プログラムのデバッグ 155
—MSX-S BUGを使う—

7章 応用プログラミング 169
—MSX固有の機能を引き出す—

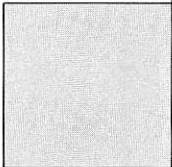
本文イラスト■三善和彦

1章

アセンブラを使うための基礎知識

―アセンブラとアセンブリ言語とMSX-DOS―

皆さんはマシン語についてどのようなイメージを持っているでしょうか？　何の意味もないように見える16進数の羅列を思い出してゾッとする人もいるでしょう．しかし，実際にマシン語のプログラムを作るときには，**「アセンブラ」**という道具を使うため，16進数の列は必要ありません．アセンブラは，マシン語の代わりにもっとわかりやすい言葉（**アセンブリ言語**）でプログラミングすることを可能とするもので，マシン語プログラム開発時の必需品といえるでしょう．

本章では，まずアセンブラがどんな道具であるかを説明します．そして，具体的なアセンブラの仕組みを調べてみます．その方法としては，アセンブリ言語のプログラムとマシン語，あるいは他の言語のプログラムとを比較します．また，プログラムの開発をいろいろな面で支えてくれる“MSX-DOS”が何を目的に作られたものなのか，どんな機能を持っているのか，プログラムを作るときにどのように役立つのかなどについても考えてみましょう．

1-1 アセンブラとアセンブリ言語

マシン語の本を読むと,「マシン語のプログラムを作るには,アセンブリ言語でプログラムを書いて,アセンブラという道具を使う」などと書かれていますが,ピンとこない人もいるでしょう.そこでまず,アセンブラという道具が何をしてくれるのか,アセンブリ言語がどんな言語なのかなどの疑問に答えていくことにします.

アセンブラとは?

マシン語はCPUが直接理解できる唯一の言語ですが,人間にはなかなか理解することができません.このマシン語をなんとか人間にわかりやすい形に直すことができれば,プログラミング作業の効率が上がるはずです.つまり,私たちは人間にわかりやすい言語でプログラムを書き,そのプログラムをあとでマシン語に変換すればよいのです.この作業を行う道具をアセンブラ(Assembler)といいます.

別のいい方をすれば,アセンブラの働きとは,人間が書いたもとのプログラム(ソース・プログラム)をマシン語のプログラム(オブジェクト・プログラム)に変換することなのです.このマシン語への変換をアセンブル(Assemble)といいます(図1.1).

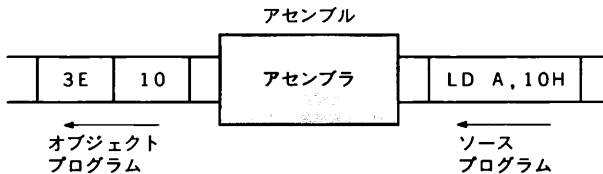



図1.1 マシン語への変換(アセンブル)

アセンブラによってマシン語に直されるソース・プログラムは，アセンブリ言語（Assembly Language）と呼ばれることばで書かれています．このアセンブリ言語はマシン語と密接な関係を持っています．たとえばマシン語がCPUの種類によって違うように，アセンブリ言語もCPUの種類によって異なるのです．私たちの使っているMSXのCPUはZ80と呼ばれるものですから，私たちはこれから，Z80のアセンブリ言語について見ていくことになります．アセンブリ言語で書かれたプログラムのなかの命令（ニーモニック）の1つ1つには，それに対応するマシン語があります．私たちは，このニーモニックを打ち込みアセンブルを行うことによって，目的のマシン語のプログラムを作ることができるのです．

アセンブリ言語と他のプログラミング言語

アセンブリ言語はプログラミング言語の1つですが，どのような特徴を持っているのでしょうか？　ここでは他の言語との比較を行うことで，アセンブリ言語の特徴を浮き彫りにします．

比較する言語は，馴染みのあるBASIC言語（紹介の必要はないでしょう）と，今流行しているC言語の2つです．C言語は1972年にAT&Tのベル研究所でミニコン用に開発された比較的新しい言語です．しかし，その後多くのプログラマーに熱烈な支持を受け，今ではMSXを含めた多くのパソコン上で使えるようになり，メジャーな言語の1つとなっています．C言語で書いたプログラムの例をリスト1.1に示します．

```
/* key code print program */

#include    <stdio.h>
main()
{
    int c;

    printf("¥nInput Char ? ");
    c = getchar();
    printf("¥nHex   Code = %x H¥n", (unsigned)c);
}
```

リスト1.1　C言語のプログラム

これらの言語とアセンブリ言語との具体的な比較の視点として，まずプログラムの実行までの手順を取り上げてみることにします．

BASIC言語ではプログラムを“RUN”させると，プログラムを1語ずつ見てその意味を解釈し，その意味に従って実行していきます．このプログラムの実行は図1.2のようになります．

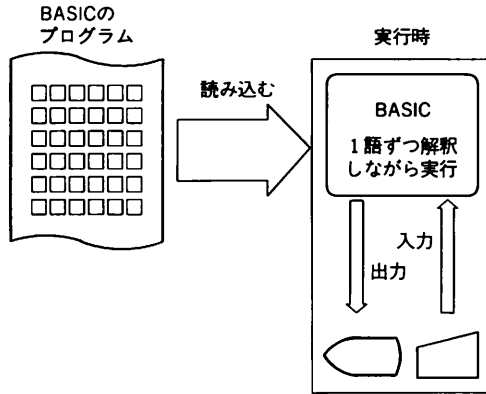



図1.2　BASICプログラムの実行

BASICでは，実行時に1行ごとにプログラムを解釈していきます．これが，BASICのプログラムが遅いといわれる理由なのです．しかし，実行時にプログラムを解釈するために，プログラムの間違いを指摘してエラーメッセージを出すことができるというメリットもあります．BASICのように実行時にプログラムを解釈して実行する方法を「**インタープリタ**」(Interpreter)といいます．

いっぽう，アセンブリ言語の場合はどうなのでしょうか？　さきほども説明したようにプログラムを実行させるためには，あらかじめアセンブルという作業が必要です．プログラムに間違いがあるかどうかはこのときに調べられます．そのため，プログラムの実行は非常に速くなるわけです．しかし，プログラムの間違いがあらかじめわかるといっても，プログラムの論理的な間違いはいっさい調べられません．ですから，この種の誤りを含んでいると実行時にわけのわからない結果を返したり，何も反応しなくなる状態（いわ

ゆる暴走）になってしまいます．BASICでは，このような間違いでは CTRL ＋ STOP で実行を止めることができ，変数の値を調べたりしてプログラムを修正することができました．しかし，マシン語（アセンブリ言語のプログラムをアセンブルして得られる）プログラムではキーを受け付けなくなってしまうため，リセット・ボタンを押す以外に実行を止める方法はありません．この場合，実行していたプログラムは失われてしまいます．

では，C言語の場合を見てみましょう．C言語は，普通「コンパイラ」(Compiler)と呼ばれる処理系でマシン語に変換されます．その実行までの流れは次のようになっています．

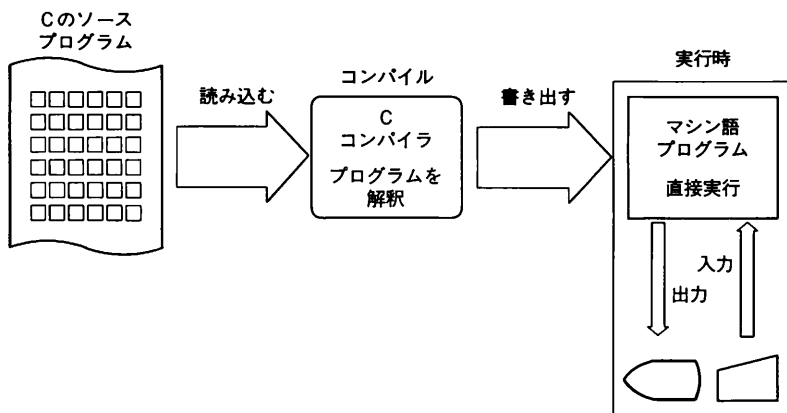



図1.3　C言語によるプログラムの実行

コンパイラ言語ではインタープリタ言語と違って，実行されるのはれっきとしたマシン語ですから暴走の危険は残っています．しかし，その代わりに，比較的高速な実行が期待できるのです．「比較的高速」ということばを使ったのは，どんなに優れたコンパイラでも，熟練したプログラマが気合いを入れて作ったアセンブリ言語のプログラムより速いものは作れないからです．それでも，BASICなどのインタープリタ言語に比べると実行速度がずっと速いことはいうまでもありません．

次にもう1つ別の視点として，プログラムのなかの命令語を比較してみましょう．

前にも述べたようにアセンブリ言語では命令語（ニーモニック）がマシン語コードに1対1で対応しています．したがって，CPUの持つすべての機能が利用できます．こういう視点で見ると，**「アセンブリ言語＝マシン語」**といっても差し支えありません．

アセンブリ言語以外の言語では，インタープリタ／コンパイラ言語に関わらず，命令語1語は，マシン語の命令1語に置き換えることはできません．事実上これらの言語の1命令はマシン語のサブルーチンに相当するといってもよいくらいです．逆にこれらの言語（いわゆる高級言語）の命令は，そのCPUに依存して決められたものでないため，機種やCPUに依存しないプログラムを書くことができます．

プログラミング言語にもいろいろな種類があり，それぞれに特徴があります．最後にこれまでできたプログラミング言語の特徴をまとめておきます（表1.1）．

言　語	特　　徴
BASIC （インタープリタ）	・実行速度が遅い ・初心者にもわかりやすい ・手軽にプログラミングできる
C （コンパイラ）	・実行速度は比較的高速 ・プログラムの他機種への移殖性が高い ・大規模プログラムのプログラミングが容易
アセンブリ言語	・実行速度がきわめて高速 ・ハードウェアに密着したプログラミングが可能 ・プログラムの実行形式ファイルがコンパクト

表1.1　プログラミング言語とその特徴

1.2 アセンブリ言語の概要

ここではアセンブリ言語について、サンプル・プログラムを見ながら説明していきます．ここで使うサンプル・プログラムの処理内容は「キーボードから1文字入力すると、その文字のアスキーコードを16進数で表示する」というものです．BASICでこの処理を記述するとリスト1.2のようになります．

```
10 ' key code print program
20 PRINT "Input Char ? ";
30 C$ = INPUT$(1)
40 PRINT
50 PRINT "Hex   Char = ";
60 PRINT HEX$(ASC(C$)); "H"
70 END
```

リスト1.2 BASICのプログラム

この処理内容をアセンブリ言語で記述するとどうなるでしょうか．ここではソース・リストを見るのではなく，「アセンブル・リスト」と呼ばれるリストを見ることにします．アセンブル・リストにはマシン語のオブジェクトとアセンブリ言語のソース・プログラムが並べて書かれており，プログラムがどのようにアセンブルされたかがひと目でわかるようになっています(リスト1.3)．

ソース・プログラム
マシン語のオブジェクト
ロードアドレス

```
                            ;***** PRINT KEY CODE IN HEXADECIMAL *****

                                    .Z80                ; We use Zilog codes

000D                        CR      EQU     0DH         ; Carrige Return
000A                        LF      EQU     0AH         ; Line Feed
0005                        SYSTEM  EQU     0005H       ; System Call Entry

0000'                               ASEG                ; Absolute SEGment
                                    ORG     100H        ; Set Start Address

                            ;--- Main Routine ---

0100    11 011D                     LD      DE,MSG1     ; Print MSG1
0103    CD 0141                     CALL    PUTMSG
0106    0E 01                       LD      C,1         ; [Console Input]
0108    CD 0005                     CALL    SYSTEM
010B    F5                          PUSH    AF          ; Save Character
010C    11 012D                     LD      DE,MSG2     ; Print MSG2
010F    CD 0141                     CALL    PUTMSG
0112    F1                          POP     AF          ; Get Character
0113    CD 0147                     CALL    PUTHEX      ; Print code in Hex
0116    11 013D                     LD      DE,MSG3     ; Print MSG3
0119    CD 0141                     CALL    PUTMSG
011C    C9                          RET                 ; Return to MSX-DOS

011D    0D 0A 49            MSG1:   DB      CR,LF,'Input Char ? $'
0120    6E 70 75
0123    74 20 43
0126    68 61 72
0129    20 3F 20
012C    24
012D    0D 0A 48            MSG2:   DB      CR,LF,'Hex   Code = $'
0130    65 78 20
0133    20 20 43
0136    6F 64 65
0139    20 3D 20
013C    24
013D    48 0D 0A            MSG3:   DB      'H',CR,LF,'$'
0140    24

                            ;--- Print Message (DE = Pointer) ---
0141                        PUTMSG:
0141    0E 09                       LD      C,9         ; [String Out]
0143    CD 0005                     CALL    SYSTEM
0146    C9                          RET                 ; MSGOUT end
```

```
                 ;--- Print Acc in Hexadecimal Form ---
0147             PUTHEX:
0147  F5                 PUSH   AF       ; Save Code
0148  0F                 RRCA            ; Shift Code
0149  0F                 RRCA            ;  4bits(high)
014A  0F                 RRCA            ;     <->
014B  0F                 RRCA            ;  4bits(low)
014C  CD 0150            CALL   HEXSUB   ; Print 4bits
014F  F1                 POP    AF       ; Restore Code

0150             HEXSUB:                 ;[ Print 4bits(low) ]
0150  E6 0F              AND    0FH      ; Use 4bit(low) only
0152  C6 30              ADD    A,'0'    ; Convert
0154  FE 3A              CP     '9'+1    ;   0~ 9 => '0'~'9'
0156  38 02              JR     C,HEXS1  ;  10~15 => 'A'~'F'
0158  C6 07              ADD    A,'A'-10-'0'
015A             HEXS1:
015A  5F                 LD     E,A      ; E = Out Code
015B  0E 02              LD     C,2      ; [Console Out]
015D  CD 0005            CALL   SYSTEM
0160  C9                 RET             ; PUTHEX/HEXSUB End

                         END             ; Program End
```

リスト1.3　アセンブル・リスト

では，このアセンブル・リストを用いてアセンブリ言語のプログラムのそれぞれの要素がどのような働きを持っているのかを説明していきましょう．

ニーモニックと擬似命令

ソース・プログラムは行を単位として分けられますが，セミコロン（；）より右側はコメントを入れる部分（BASICのREM文に相当）ですから，アセンブル結果には影響を与えません．アセンブルされるのは各行のセミコロンより左側だけです．

Z80の命令語（ニーモニック）のある行には，その命令語に対応する1から3バイト程度のマシン語の命令に変換され，プログラムが置かれるメモリ上の位置（ロードアドレス）とともに書き込まれています．これらの命令はCPUが実行できるマシン語コードに直されています．

次に，アセンブリ言語のプログラムのなかで「**擬似命令**」と呼ばれる命令について注目します．この擬似命令は，アセンブラに対してアセンブルの指示を与えるものなのです．リスト 1.3 のプログラムに出てくる擬似命令の意味と使い方については 3 章で詳しく説明することにして，ここでは簡単にその内容を整理しておきましょう．

- ORG … アセンブルするアドレスを決める
- ASEG … アセンブル時にそのアドレスを決めるように指定
- .Z80 … アセンブルするニーモニックが Z80 のものであると宣言する
- EQU … シンボル（後述）の値を決める
- DB … プログラム中のデータ領域の内容を決める
- END … アセンブルするプログラムの終わりを示す

なお，擬似命令はアセンブラによって異なり，ここにあげた擬似命令は MSX·M-80 というアセンブラのものです．また，本書でこれ以降出てくるプログラムも MSX·M-80 でアセンブルされることを前提にしています．なお，MSX·M-80 については 2 章で詳しく紹介します．

シンボルとラベル

アセンブリ言語のプログラムを構成する要素としてもう 1 つ重要なものが「**シンボル**」(Symbol) です．リスト 1.3 のサンプル·プログラムで用いられているシンボルは次のものです．

CR, LF, SYSTEM, PUTMSG, PUTHEX, HEXSUB, HEXS1, MSG1, MSG2, MSG3

シンボルは，特定の数値を名前（**シンボル名**）で表すものです．アセンブラはシンボルが出てくると，そのシンボルの意味する数値に変換してアセンブルを行うのです．たとえば，ソース·プログラムで，

```
SYSTEM    EQU     0005H
```

としているのは，「SYSTEM」というシンボルに 0005H（16進数で5）という値を定義しているのです．ですから，

```
CALL      SYSTEM
```

という命令があると，

```
CALL      0005H
```

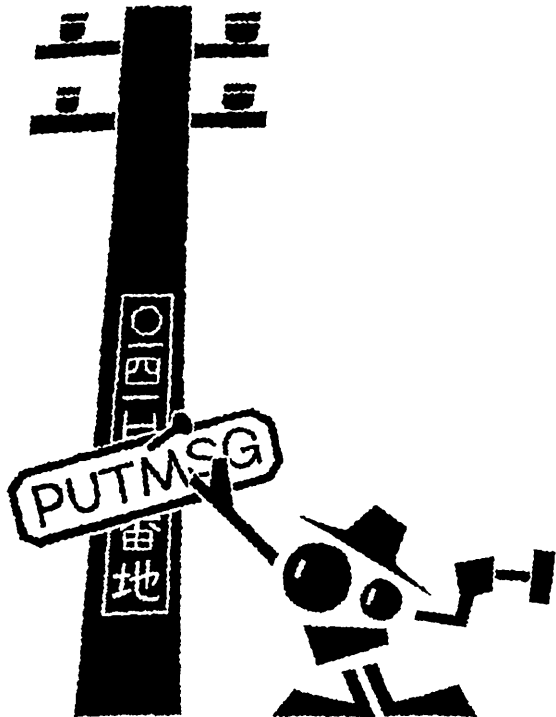
のようにアセンブラの内部でシンボルを数値に変換してくれるのです．シンボルを用いる利点は，プログラムのなかで私たちにとって意味を持たない数値をなんらかの意味を持たせた文字列にして表せることです．その結果として，プログラムが書きやすく読みやすいものとなるのです．

アセンブラがシンボルを数値に変換するためには，そのシンボルの示す数値をソース・プログラムのなかで定義しておかなければなりません．擬似命令 EQU による方法もその1つですが，コロン（：）を使って定義する場合もあります．リスト1.3のなかで（：）を用いて定義されているシンボルは次のものです．

```
PUTMSG, PUTHEX, HEXSUB, HEXS1, MSG1, MSG2, MSG3
```

この種類のシンボルのことを，「ラベル」(Label) といいます．アセンブル・リストを見ればわかりますが，ラベルの示す値はそのメモリ上のアドレスを示しています．たとえば，「PUTMSG」は 0141H 番地を示しているといった具合いです．ラベルを用いることで，プログラマはソース・プログラムを書く段階にアドレス値を計算する必要はなくなります．ただ単にラベルを付けて，そのラベルを参照するように書いておけばよいのです．アセンブラは，プログラムのロードアドレスを計算してラベルをアドレス値に直してくれるのです．ラベルが使えるということはたいへん便利なことで，この点だけを取っ

てもアセンブラを使う価値があります．

ここでは，アセンブリ言語のプログラムを調べることで，アセンブラの働きについてざっと見てきました．以上のように，アセンブラはマシン語プログラムの開発にはなくてはならないプログラムなのです．このようにプログラム開発などの役に立ち，よく使われるプログラムは，いわばソフトウェアの「道具」（ソフトウェア・ツール）であるということができます．アセンブラもその道具の1つです．

1-3 MSX-DOSとプログラミング

本書で用いるアセンブラ MSX・M-80 を使うには，MSX-DOS という OS（**オペレーティング・システム**）が必要です．OS はアセンブラなどのソフトウェア・ツールの実行やその処理の下請けをする基本プログラムなのです．ここでは，アセンブラやアセンブリ言語からは少し離れて，MSX-DOS という OS の働きについて考えることにしましょう．それは OS についての知識が，これから私たちが作るプログラムにも密接に関わってくるからです．

一般に OS の基本的な機能としては次のようなものが考えられます．

- **周辺機器とのデータの入出力**
- **ファイル管理**
- **プログラムの実行**

MSX-DOS はもちろんこれらの機能を持っています．ここでは，それぞれの機能が MSX-DOS でどのようになっているのかを見ておきましょう．

データの入出力

●システムコールとは？

コンピュータには周辺機器が不可欠なものです．周辺機器がなければ，コンピュータに何を実行させるのかの命令を与えることもできませんし，コンピュータの出力した結果を私たちが知ることもできません．MSX-DOS がサポートしている周辺機器は次のとおりです．

・キーボード
・ディスプレイ
・プリンタ
・ディスク・ドライブ　（最大8台まで，順にドライブA，ドライブB，……，ドライブH）

これらの周辺機器に対してそれぞれのプログラムが入出力のための複雑なサブルーチンを書くことは，プログラマにとっても大きな負担になります．この複雑な手順となる入出力についてはOSがあらかじめ標準的なルーチン群を用意してあり，これらのルーチンを利用すれば，プログラムを効率よく書くことができるのです．MSX-DOSの持つ入出力のルーチンは「システムコール」(System Call) という形で統一して呼び出すことができます．

入出力の操作にシステムコールだけを使ったプログラムは，他の機種への移植が簡単になります．MSX-DOSのシステムコールは，その呼び出し方や基本的な機能は，CP/M-80という8ビットパソコンの標準的なOSのBDOSコールと互換性を持っており，一部の機能はさらに強力なものへと拡張されています．したがって互換性のある部分を用いれば，MSX-DOS上でCP/M-80のプログラムを書いたりその逆も可能です．

●システムコールの呼び出し方

システムコールは，プログラムのなかでZ80 CPUのCレジスタにファンクション番号を入れ，0005H番地を呼び出すことによって実行されます．サンプルとして取り上げたプログラム（リスト1.3）のなかで，次のように書いてある行は，実はシステムコールを行うという意味だったのです（シンボルSYSTEMには0005Hが定義されている）．

```
CALL    SYSTEM
```

また，システムコールの種類によっては，呼び出すときにレジスタやメモリに値を設定しておく必要があります．さらに，システムコールで得られた値はレジスタやメモリを介してプログラムに返されます．なお，システムコー

ルの前後では，受け渡しに使っていないレジスタでも破壊されてしまうことがあるので，必要があればそれらのレジスタを保存しておかなくてはなりません．システムコールを行うときの実際の手順についてはリスト1.3を見て参考にしてもらえばよいのですが，この手順をまとめると次のようになります．

- 必要があればレジスタを保存する
- レジスタやメモリに必要な値を設定する
- Cレジスタにファンクション番号を入れる
- 0005H 番地をコールする
- システムコールからの戻り値を取り出す
- 保存していたレジスタの内容をもとに戻す

●システムコールの種類

MSX-DOSのシステムコールは，表1.2に示すように全部で42個用意されています．

なお，それぞれのシステムコールの具体的な使い方については巻末のAppendixを参照してください．

ファイル管理

ディスク・ドライブはパーソナルコンピュータの周辺機器としては最も汎用性のある外部記憶装置です．その長所は大量のデータを高速に任意の順序で受け渡すことができる点です．このため，OSのなかでも主にディスク・ドライブを管理するように作られたものを“DOS”（ディスク・オペレーティング・システム）と呼びます．名前からもわかるようにMSX-DOSもDOSの1つです．なお，ディスクの使えるOSをDOSと呼ぶのはパーソナルコンピュータの場合だけです．もっと大型のコンピュータ（ミニコン以上）ではディスク装置を管理することができるオペレーティング・システムでも単にOSと呼ばれています．

ファンクションNo.	機　能	ファンクションNo.	機　能
00H	システムリセット	14H	シーケンシャルなファイルの読み出し
01H	コンソールから1文字入力(入力待ちあり，エコーバックあり，コントロール・コードチェックあり)	15H	シーケンシャルなファイルの書き出し
		16H	ファイルの作成
02H	コンソールへ1文字出力	17H	ファイル名の変更
03H	補助入力装置から1文字入力	18H	ログイン・ベクトルの獲得
04H	補助出力装置へ1文字出力	19H	デフォルト・ドライブ番号の獲得
05H	プリンタへ1文字出力	1AH	DTAの設定
06H	コンソールから1文字入力(入力待ちなし，エコーバックなし，コントロール・コード・チェックなし)／1文字出力	1BH	ディスク情報の獲得
		1CH ～ 20H	無効
07H	コンソールから1文字入力(入力待ちあり，エコーバックなし，コントロール・コード・チェックなし)	21H	ランダムなファイルの読み出し
		22H	ランダムなファイルの書き込み
08H	コンソールから1文字入力(入力待ちあり，エコーバックなし，コントロール・コード・チェックあり)	23H	ファイルサイズの獲得
		24H	ランダムレコード・フィールドの設定
09H	文字列出力	25H	無効
0AH	文字列入力	26H	ランダム・ブロック書き込み
0BH	コンソールの状態チェック	27H	ランダム・ブロック読み出し
0CH	バージョン番号の獲得	28H	ランダムなファイルの書き込み(不用部を00Hで埋める)
0DH	ディスクリセット	29H	無効
0EH	デフォルト・ドライブの設定	2AH	日付の獲得
0FH	ファイルのオープン	2BH	日付の設定
10H	ファイルのクローズ	2CH	時刻の獲得
11H	ワイルドカードに一致する最初のファイルの検索	2DH	時刻の設定
12H	ワイルドカードに一致する2番目以降のファイルの検索	2EH	ベリファイ・フラグの設定
		2FH	論理セクタを用いた読み出し
13H	ファイルの抹消	30H	論理セクタを用いた書き込み

表1.2　システムコール一覧

DOSはディスクに対するデータの入出力を「ファイル」(file)という単位に分けて保存し，データがいつでも取り出せるように管理しています(図1.4)．

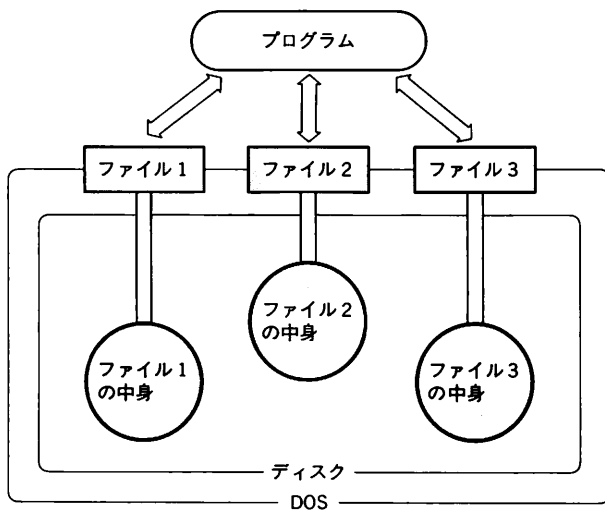



図1.4 ディスクをファイル単位で管理

このファイルの1つ1つのなかには，意味のあるデータが1まとまりとなって入れてあり，データをいつでも取り出すことができるようになっています．それぞれのファイルには「**ファイル名**」という名前が付き，これにより目的のファイルと他のファイルとが区別されます．

DOSはファイルを管理する基本プログラムを持っていますから，ユーザー・プログラムのなかから簡単にファイルを扱えるようになっています．それだけではなく，私たちがファイルの操作を直接行えるような命令（**内部コマンド**）も持っています．MSX-DOSでは，**表1.3**に示す内部コマンドがあります．

なお，内部コマンドの具体的な使い方や詳しい説明については「MSX-DOS入門」（(株)アスキー発行）などを参照してください．

BASIC	ディスクBASICの起動
COPY	ファイルのコピー
DATE	日付の表示，変更
DEL(ERASE)	ファイルの削除
DIR	ファイル名の表示
FORMAT	フォーマット
MODE	行の表示幅変更
PAUSE	バッチコマンドの一時停止
REM	バッチコマンドのコメント
REN(RENAME)	ファイル名の変更
TIME	時間の表示，変更
TYPE	ファイルの内容を表示
VERIFY	ディスクへの書き込みチェックのON／OFF

表 1.3　MSX-DOS の内部コマンド一覧

プログラムの実行

●外部コマンド

MSX-DOS はデータをファイルとして管理していますが，データだけでなくプログラム (ユーザー・プログラム) もファイルとして管理しています．このユーザー・プログラムのはいったファイルには，ファイル名に拡張子として“.COM”が付けられています．そのため，実行可能なプログラム・ファイルのことを「COM 形式のファイル」と呼ぶことがあります．

MSX-DOS 上では，ユーザー・プログラムは外部コマンドとして呼び出され，実行されます．外部コマンドの一般的な呼び出し方は，次のようになっています．

[〈ドライブ名〉：] 〈コマンド名〉　[引数……]

ここで 〈コマンド名〉 には，拡張子を除いたファイル名を指定します．また，〈ドライブ名〉 を省略した場合にはデフォルト・ドライブの指定があったものとみなされます．たとえばデフォルト・ドライブに存在する “GVBGN.COM” というファイルを実行するには，“GVBGN” と入力します．

●プログラムの実行まで

私たちがプログラムを実行しようとしてコマンド名を入力すると，DOS はそのプログラムをメモリ上に読み込み，そのプログラムに制御を移す（そのプログラムが実行される）のです．MSX-DOS では，プログラムが読み込まれる先頭のアドレスおよび実行開始アドレスは 100H 番地です．もうおわかりでしょうが，リスト1.3のサンプル・プログラムのなかで

```
ORG     100H
```

としていたのはこのためだったのです．

MSX-DOS 上でユーザー・プログラムが実行されるまでのようすは図 1.5 のようになります．

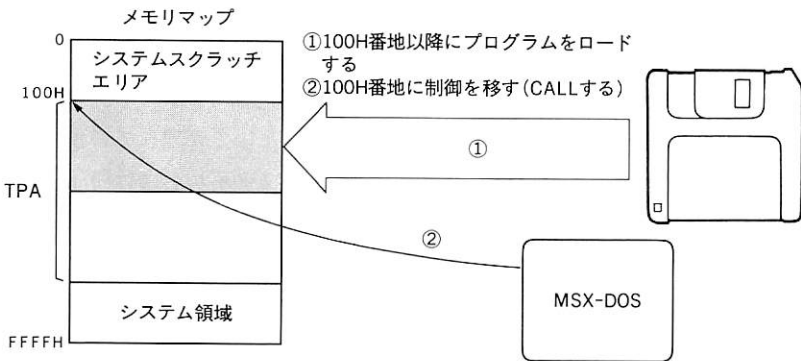



図 1.5 プログラムの実行

なお，この図ではメモリ空間が 3 つに分けられています．それぞれの意味はおよそ次のようになっています．

・システムスクラッチエリア

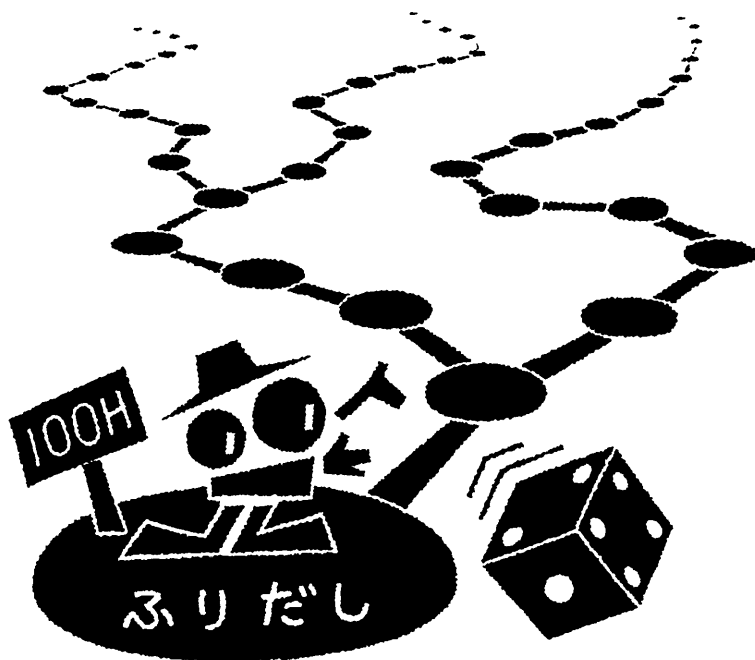
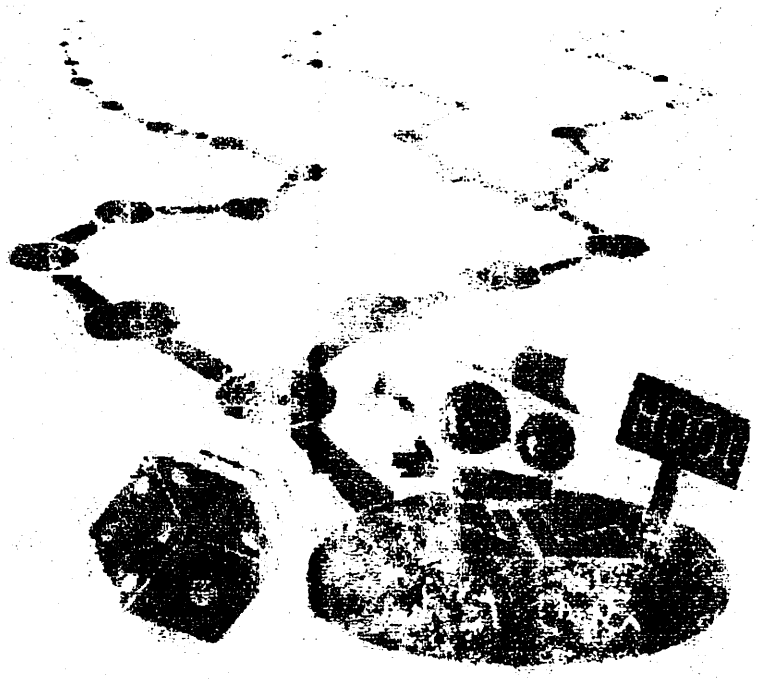
0H 番地から 0FFH 番地までの領域で，システム（DOS）とユーザー・プログラムの橋渡しをする領域といえます．システムコールの飛びさきもこの領域にあります．また，5 章で詳しく説明しますが，コマンドに渡す引数の内容

もこの領域に格納されます.

・TPA (Transient Program Area)

100H 番地から始まり, 6~7 番地の内容が示しているアドレスの1バイト前までがTPA (Transient Program Area) と呼ばれる領域です. この領域はユーザー・プログラムが自由に使用できます. なお, 6~7 番地はシステムコールの飛びさきを示しています.

・システム領域

TPA より高位のアドレス領域はシステムが使用している領域であり, プログラムが勝手に使ってはいけません.

2章

プログラム開発の手順

—M-80／L-80の使い方—

1章ではアセンブラやMSX-DOSの働きについて話をしてきましたが，これらを具体的にどのように使うのかということについては，何も触れていませんでした．これまでの話はプログラムを組むために必要な予備知識だったのですが，早くプログラムを組みたいと思っている人も多いことでしょう．

しかし，プログラムを組むためには，いくつかのソフトウェア・ツールが必要となることを思い出してください．具体的には，アセンブリ言語でプログラムを記述するための**エディタ**（Editor），そしてもちろんアセンブルを行うための**アセンブラ**（Assembler），さらにアセンブルされたファイルをまとめて，実行可能なCOM形式のファイルに変換するための**リンク・ローダ**（Link Loader）といったものがあげられます．つまりプログラムを組むためには，これらのツールの使い方を知る必要があるのです．

そこで本章では，まずプログラムを組むために必要なツールが含まれた“MSX-DOS TOOLS”というソフトウェアを紹介します．そのあと，これらのソフトウェア・ツールを使ってサンプル・プログラム（1章で紹介したもの）を入力／アセンブル／リンク・ロードし，できあがったプログラムを実際に実行してみることにします．

2-1 MSX-DOS TOOLS

“MSX-DOS TOOLS”は，その名のとおり，MSX-DOSの上で本格的なプログラム開発をするためには欠かすことのできない各種のコマンド（プログラム）を集めたものです（写真2.1）.

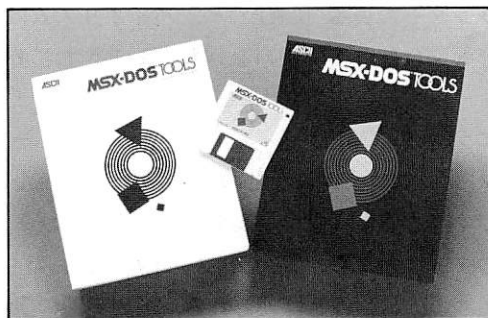



写真2.1　MSX-DOS TOOLS

この“MSX-DOS TOOLS”に収められているプログラムを，その働きによって分類すると次のようになります.

- ・MSX-DOS
- ・MSX-DOSをより便利に使うためのコマンド（28種類）
- ・テキスト・ファイルを作り，編集するためのスクリーン・エディタ
- ・アセンブリ言語のソース・プログラムをマシン語にするアセンブラ，リンク・ローダなどのユーティリティ

では，これらについて順に見ていくことにしましょう.

● MSX-DOS

ここでいうMSX-DOSとは，MSX-DOSを実行するために必要な2つのファイル，つまり“MSXDOS.SYS”と“COMMAND.COM”のことです．この2つのファイルがないとMSX-DOSは動きません．

●コマンド（28種類）

MSX-DOSをより便利に使うためのコマンド群です．これらのコマンドはCOM形式のファイルになっており，外部コマンドとして実行されます．このなかには，ディスクの状態表示（CHKDSK）や，ファイル内容の16進表示（DUMP）など，DOSの操作やプログラミングを補助してくれるコマンドもありますが，カレンダーの表示（CAL）や，バイオリズムの表示（BIO）のようにプログラミングには直接関係のないものまでさまざまです（表2.1）．

●スクリーン・エディタ“MED”

“MED”は，ソース・プログラムなどのテキスト・ファイルを入力／編集するためのツールです．このようなツールをエディタと呼びます．エディタはプログラムを入力する以外にも，データ・ファイルを作ったりバッチ・ファイルを作ったりというようにしばしば使われる，応用範囲の広いツールです．

エディタは大きく2種類に分けられます．ある指定した行について編集していくポインタ形式と，画面上に表示されている文章をカーソルを動かして自由に編集できるスクリーン・エディット形式です．現在ではスクリーン・エディット形式の方が主流になっていますが，その理由はスクリーン・エディット形式の方が自由度が高く，高機能なためです．もちろん“MED”はスクリーン・エディット形式です．

“MED”はテキストの編集をサポートするためにさまざまな機能を持っています．こういうと「使いこなすのは大変なのでは」と心配になるかもしれません．しかし，最初からすべての機能を使う必要はないので，とりあえず入力／編集に必要な命令を覚えておき，基本的な操作に慣れてからいろいろな命令を覚えていけばよいのです．

コマンド名	機　　能
BEEP	BEEP音の発生
BIO	バイオリズムの出力
BODY	ファイルの一部の切出し
BSAVE	HEXファイルのバイナリ・ファイルへの変換
CAL	カレンダーの表示
CALC	簡易電卓
CHKDSK	ディスクの状態の表示
CLS	クリアスクリーン
DISKCOPY	バックアップコピー及び照合
DUMP	ファイルの16進ダンプ
ECHO	コマンド行の表示
EXPAND	スペース／タブの変換
GREP	任意の文字列の検索
HEAD	ファイルの先頭部分の表示
HELP	コマンドリファレンスの表示
KEY	ファンクションキーの設定
LIST	BASICの中間言語ファイルのソース・ファイルへの変換
LS	ディレクトリの詳細情報の表示
MENU	ディレクトリのファイルの諸操作
MORE	ファイルの表示
PATCH	バイナリ・ファイルの更新
SLEEP	アラーム機能
SORT	ソート(クイックソート)
TAIL	ファイルの末尾部分の表示
TR	文字列の置き換え
UNIQ	重復行の削除
VIEW	ファイルの表示(スクロール)
WC	語数，行数，ページ数カウント

表 2.1　MSX-DOS TOOLS コマンド一覧

●アセンブラ，リンク・ローダなどのユーティリティ

プログラミング，とくにアセンブラによるプログラミングをサポートするのがこれらのユーティリティです．ユーティリティには次のものがあります．

- MSX・M-80 …… マクロ・アセンブラ
- MSX・L-80 …… リンク・ローダ
- CREF-80 …… クロス・リファレンサ
- LIB-80 ……… ライブラリ・マネージャ

アセンブリ言語をマシン語のプログラムにするだけならアセンブラだけあればよいのでは，と思っている人も多いかもしれません．しかし"MSX-DOS TOOLS"に含まれるアセンブラは実行可能なマシン語のプログラムを直接出力しません．まず，アセンブラでソース・プログラムを読み込んでオブジェクト・ファイル（".REL"という拡張子が付く）を作ります．次にリンク・ローダでオブジェクト・ファイルを読み込んで，マシン語の実行ファイル（COM形式）を作成します．このようにソース・プログラムからマシン語プログラムを作成するためには，2段階の手順を必要とするのです（図2.1）．

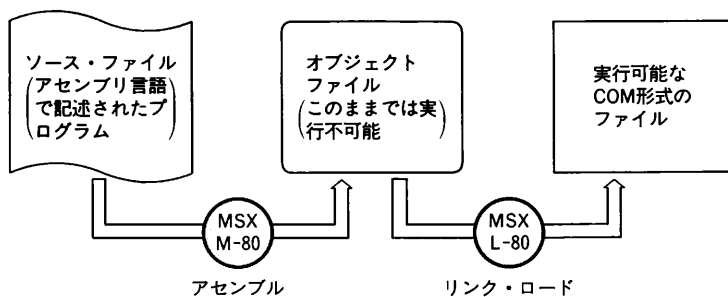



図2.1 アセンブリ言語のプログラムをマシン語にする

「どうしてこんなに面倒な」と思われるでしょう．あえてこのような手順をふむのはモジュール別のプログラム開発を可能にするためなのです．この方法については4章で詳しく説明しますが，簡単にいえばプログラムを機能的にまとまった単位（モジュール）ごとに別々に作成して，あとでまとめて目的のプログラムを得るというものです．この方法によるメリットはとくに大規模なプログラムを作成するときに現れます．

また，本書では触れませんが，ユーティリティに含まれているクロス・リファレンサやライブラリ・マネージャはこのモジュール開発を助けてくれるツールなのです．

2-2 サンプル・プログラムの入力

では，この"MSX-DOS TOOLS"を使って，実行可能なCOM形式のファイルを作ってみることにしましょう．ここで取り上げるサンプル・プログラムは1章で紹介したものです．ここにもう一度示しておきましょう(リスト2.1)．

```
;***** PRINT KEY CODE IN HEXADECIMAL *****

        .Z80                    ; We use Zilog codes

CR      EQU     0DH             ; Carrige Return
LF      EQU     0AH             ; Line Feed
SYSTEM  EQU     0005H           ; System Call Entry

        ASEG                    ; Absolute SEGment
        ORG     100H            ; Set Start Address

;--- Main Routine ---

        LD      DE,MSG1         ; Print MSG1
        CALL    PUTMSG
        LD      C,1             ; [Console Input]
        CALL    SYSTEM
        PUSH    AF              ; Save Character
        LD      DE,MSG2         ; Print MSG2
        CALL    PUTMSG
        POP     AF              ; Get Character
        CALL    PUTHEX          ; Print code in Hex
        LD      DE,MSG3         ; Print MSG3
        CALL    PUTMSG
        RET                     ; Return to MSX-DOS

MSG1:   DB      CR,LF,'Input Char ? $'
MSG2:   DB      CR,LF,'Hex   Code = $'
MSG3:   DB      'H',CR,LF,'$'
```

```
;--- Print Message (DE = Pointer) ---
PUTMSG:
        LD      C,9      ; [String Out]
        CALL    SYSTEM
        RET              ; MSGOUT end

;--- Print Acc in Hexadecimal Form ---
PUTHEX:
        PUSH    AF       ; Save Code
        RRCA             ; Shift Code
        RRCA             ;  4bits(high)
        RRCA             ;     <->
        RRCA             ;  4bits(low)
        CALL    HEXSUB   ; Print 4bits
        POP     AF       ; Restore Code
HEXSUB:                  ;[ Print 4bits(low) ]
        AND     ØFH      ; Use 4bit(low) only
        ADD     A,'Ø'    ; Convert
        CP      '9'+1    ;   Ø¯ 9 => 'Ø'¯'9'
        JR      C,HEXS1  ;  1Ø¯15 => 'A'¯'F'
        ADD     A,'A'-1Ø-'Ø'
HEXS1:
        LD      E,A      ; E = Out Code
        LD      C,2      ; [Console Out]
        CALL    SYSTEM
        RET              ; PUTHEX/HEXSUB End

        END              ; Program End
```

リスト 2.1　入力された文字のアスキーコードを表示するプログラム

作業の手順は，まずこのプログラムを入力して，アセンブル／リンク・ロードし，できあがったプログラムを実行するというものです．なお，本書ではMSX にディスク・ドライブが 2 台 (A:と B:) 接続されているものとし，各ドライブには次に示すディスクがはいっているものとして話を進めていきます．

A：システム・ディスク　……　“MSX-DOS TOOLS” のディスク
B：ワーク・ディスク　……　プログラムを作るディスク

しかし，1ドライブしか接続されていない場合にこのようなディスクの振り分けだと，ファイル操作のたびにディスクの交換が必要となるために非常に面倒です(2ドライブ・エミュレーション機能を使うため)．ですから“MSX-DOS TOOLS”のうち以下のファイルを適当な空きディスクにコピーして，そのディスク1枚でシステム・ディスクとワーク・ディスクを兼用するとよいでしょう．

```
MSXDOS.SYS
COMMAND.COM
MED.COM
MED.MES
MED.HLP
M80.COM
L80.COM
```

1ドライブしかない場合，システム・ディスクもワーク・ディスクもドライブA：にはいっているものとして扱いますので，これ以降でワーク・ディスクにはいっているファイルの“B：～”とあれば，これを“A:～”と置き換えて読み進んでください．

では，スクリーン・エディタMEDを使ってサンプル・プログラムを入力することにしましょう．ここではサンプル・プログラムのファイル名を“KEYCODE.MAC”として，MSX-DOSのコマンドライン上から次のように入力してMEDを起動します．

```
MED B:KEYCODE.MAC ⏎
```

なお，MEDを起動するときの一般的な書式は次のようになっています．

```
MED  [[<ロード・ファイル名>]  <セーブ・ファイル名>]
```

ロード・ファイル名，セーブ・ファイル名はいずれも省略することができま

す．省略して立ち上げると，図2.2の画面となりファイル名を聞いてきますから，ここで入力します．

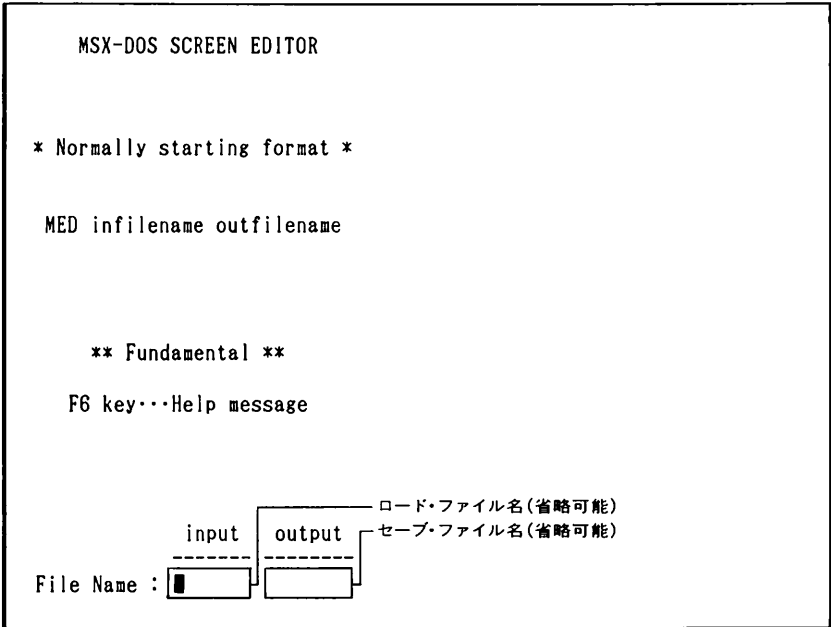



図2.2　MEDオープニングメッセージ

起動すると図2.3のような画面に変わり，テキストを入力できる状態になります．

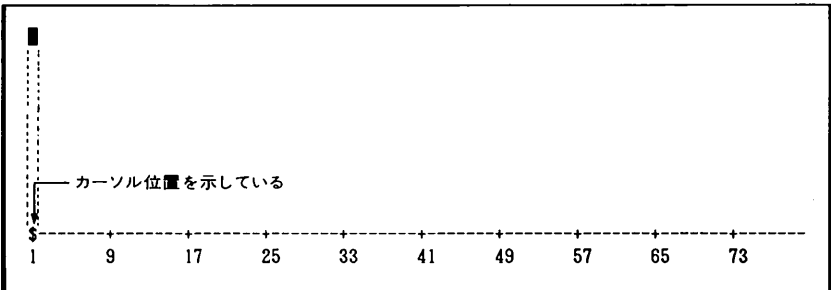



図2.3　MEDの画面　その1

画面最下行に表示されているのは，横位置(カラム数)を見るためのスケールです．スクリーン・エディタでは，現在カーソルのある位置に文字を挿入するなどの操作を行ってテキストを作成していくわけですが，このスケール上の“$”はカーソルの現在の横位置を示しているのです．ここで ESC を押すと画面は図 2.4 のようになり，各種の情報を表示します．

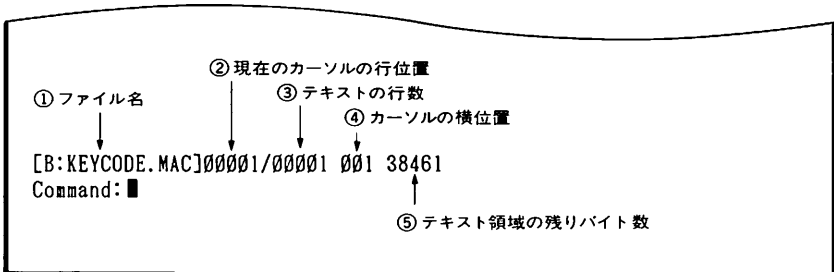



図 2.4 MED の画面 その 2

この状態で画面下から 2 行目のメッセージの意味は次のようになっています．

① 現在のファイル名
② 現在カーソルが何行目にあるかを示す．先頭行は 1
③ 読み込まれたテキストの最下行が何行目かを示す
④ カーソルの現在の横位置を示す
⑤ エディットできる空き領域の大きさ

ここで，⏎ を入力すると，また図 2.3 の画面，つまりテキストが入力できる状態に戻ります．

最初のうちはサンプル・プログラムを打ち込むことにはあまりこだわらずに，いろいろな文字を打ち込んで試してみてください．打ち込まれた文字は画面に表示されるのと同時にメモリ上のエディット・バッファと呼ばれる領域に蓄えられます．

文字の桁を揃えるためには TAB を使います．また，改行は ⏎，挿入／上書きのモード変更は INS で行います．このモードの変更によってカーソルの形が変わります．このあたりの操作は BASIC のエディタの操作と同様なのでわかりやすいでしょう．ただし，モードの変更は次に INS が押されるまで有効です．ここで特殊キーとその役割をまとめると次のようになります．

キー	役割
→	カーソルを右に移動
←	カーソルを左に移動
↑	カーソルを上に移動
↓	カーソルを下に移動
INS	挿入／上書きのモード切り換え
⏎	改行
HOME	カーソルを画面の左上隅に移動
TAB	水平タブ
BS	カーソル位置の左側の 1 文字を削除
DEL	カーソル位置の文字を削除

これらのキーも実際に入力して試してみてください．画面がいっぱいになると画面は上にスクロールしていきます．上の方を見たければ，カーソルキーで画面を動かせばよいのです．ただし，カーソルをテキストが入力されていない場所に移動させることはできません．

これらの操作で，次ページの図 2.5 のようにテキストを入力していくわけですが，これだけではエディット・バッファに蓄えられているテキストをディスクにセーブしたり，文字列の検索／置換などの機能を使うことができません．これらの機能は，エディタのコマンドとして実行することができるようになっています．このコマンドの呼び出し方は，次ページの表 2.2 のようにいくつかの種類に分けられています．

```
;***** PRINT KEY CODE IN HEXADECIMAL *****

CR      EQU     ØDH     ; Carrige Return
LF      EQU     ØAH     ; Line Feed
SYSTEM  EQU     ØØØ5H   ; System Call Entry

        .Z8Ø            ; We use Zilog codes
        ASEG            ; Absolute SEGment
        ORG     1ØØH    ; Set Start Address

;--- Main Routine ---

        LD      DE,MSG1 ; Print MSG1
        CALL    PUTMSG
        LD      C,1     ; [Console Input]
        CALL    SYSTEM
        PUSH    AF      ; Sav█

   各フィールドはTABコードで区切る(詳しくは3.1参照)

+-------+-------+-------+----$--+-------+-------+-------+-------+-------+-------
1       9       17      25      33      41      49      57      65      73
```

図 2.5　サンプル・プログラムの入力

種　類	入力形態	主な機能
ダイレクト・コマンド (直接実行コマンド)	・特殊キー, ・CTRL +アルファベットキー	カーソル移動, ロールアップ ロールダウン, 挿入, 削除 モード切換え, タブ 行の分割・結合・改行等
インダイレクト・コマンド (間接実行コマンド)	・ESC を押したあと コマンド名を入力	ファイルの読み込み, 書き出し 画面出力・プリント出力 ディレクトリ表示 フロッピーディスク諸元の表示 ヘルプファイルの表示 テキストのクリア等
ブロック・コマンド	・SELECT を押したあと コマンドを選択	ブロックのコピー, 消去, 移動, 保存

注：使用頻度の高いコマンドは, ファンクション・キーに登録されている

表 2.2　MED のコマンドの種類

ダイレクト・コマンドには2種類あります．1つはさきほどあげた特殊キーを押すもので，もう1つは CTRL （コントロールキー）とアルファベットのキーを同時に押すものです．いずれにしろそのキーを押すとすぐにコマンド

が実行されます．これに対し**インダイレクト・コマンド**を起動するには，まず ESC (エスケープキー) を押します．さきほどの図 2.4 の画面は，インダイレクト・コマンドの入力画面だったのです．ここでコマンド名を入力し，⏎を押して初めてコマンドが実行されます．このようにインダイレクト・コマンドの実行は，ダイレクト・コマンドの実行よりも多くの手順を要するようになっています．これは，テキストを破壊する可能性のあるコマンド（たとえば“NEW”，“LOAD”など）をうっかりと実行して，あとで泣きを見るのを防ぐという意味もあるのです．

これらのコマンド以外にも，テキストの一部をまとめて移動させる，**ブロック・コマンド**と呼ばれる命令もあります．このコマンドは SELECT (セレクトキー) を押し，そのブロック領域の始点，終点を指定してコマンドを入力すると実行されます（図 2.6)．

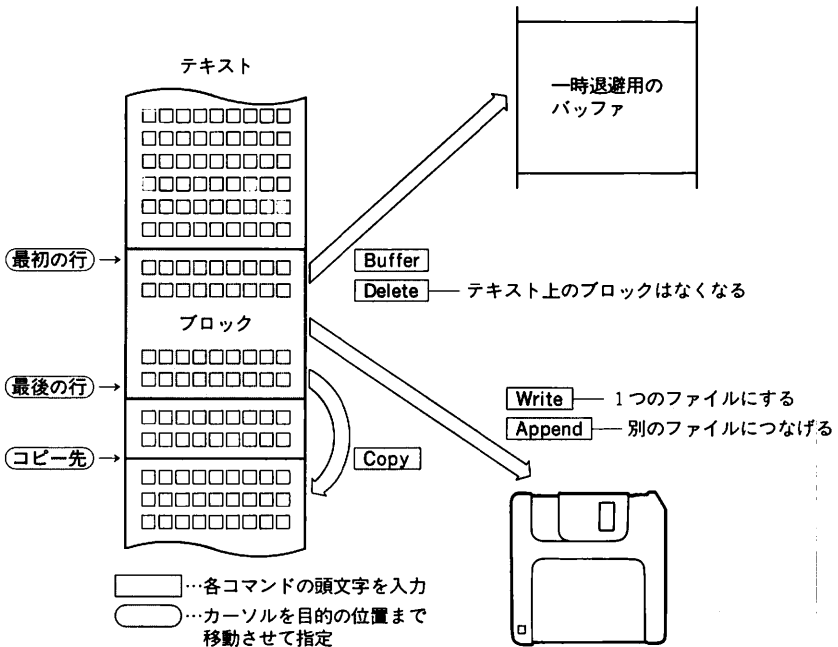



図 2.6 ブロック・コマンドの考え方

実際のコマンドとその機能については，巻末の Appendix を参照してください．またエディット中でも，[F6] を押せばコマンドの要約を見ることができますので，必要なときは参照するとよいでしょう．

前にもいいましたが，最初のうちはエディタのすべての機能を使う必要はありません．コマンドに関しても，とりあえずファイルのロード／セーブ，エディタの終了など必要最小限のものを覚えれば十分でしょう．残りのコマンドは，テキストをより速く効率的に編集する手助けとなるものなのですから，基本操作に慣れてから少しずつマスターしていけばよいのです．

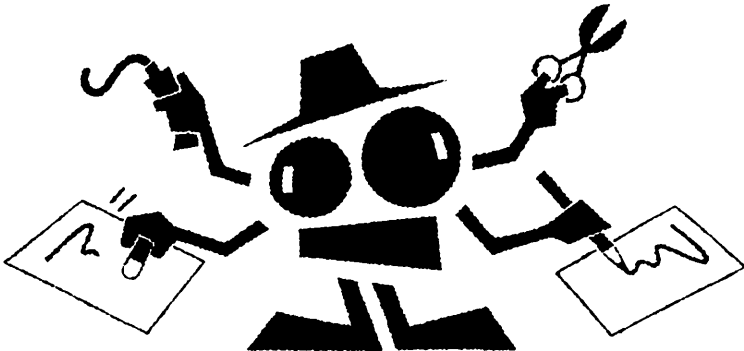
とにかく，ここでは MED を使ってサンプル・プログラムの入力が終わったものとします．エディタを終了するためには [ESC] を押して“QUIT [⏎]”と入力します．そうすると図 2.7 のように表示されます．

```
PUTHEX:
        PUSH    AF      ; Save Code
        RRCA            ; Shift Code
        RRCA            ;  4bits(high)
        RRCA            ;     <->
        RRCA            ;  4bits(low)
        CALL    HEXSUB  ; Print 4bits
        POP     AF      ; Restore Code
HEXSUB:                 ;[ Print 4bits(low) ]
        AND     ØFH     ; Use 4bit(low) only
        ADD     A,'Ø'   ; Convert
        CP      '9'+1   ;   Ø¯ 9 => 'Ø'¯'9'
        JR      C,HEXS1 ;  1Ø¯15 => 'A'¯'F'
        ADD     A,'A'-1Ø-'Ø'
HEXS1:
        LD      E,A     ; E = Out Code
        LD      C,2     ; [Console Out]
        CALL    SYSTEM
        RET             ; PUTHEX/HEXSUB End

        END             ; Program End

[B:KEYCODE.MAC]ØØØ58/ØØØ58 ØØ1 37235
Save (Y/N)?█ ……………[Y] または [⏎] でセーブされる
```

図 2.7　MED の終了

そこで[Y]（または[↵]）を押します．そうするとエディット・バッファの内容がディスクに書き込まれ，エディタが終了しDOSのコマンド待ち状態に戻ります．テキストが正しくセーブされたかどうかを，DOSのDIRコマンドやTYPEコマンドで確認しておきましょう（図2.8）．

```
A>dir b:↵
COMMAND  COM    6656 85-09-02 10:10p
MSXDOS   SYS    2432 85-08-23  9:29p
KEYCODE  MAC    1229 88-04-05  3:56p··············指定したファイル名
       3 files   717824 bytes free                でセーブされている
                                                  かを確認する
A>type b:keycode.mac ↵
;***** PRINT KEY CODE IN HEXADECIMAL *****···············ファイルの
                                                          内容が正し
       .Z80             ; We use Zilog codes              くセーブさ
                                                          れているか
                                                          を確認する
CR      EQU      0DH      ; Carrige Return
```

図2.8 作成したファイルの確認

2-3 サンプル・プログラムのアセンブル

ここではMSX・M-80(以下単にM-80と呼ぶ)の使い方について見ていきたいと思います.まずはじめに,M-80で扱うファイルについて説明しておきましょう.そのファイルとは4種類あり,それぞれの関係は図2.9のようになっています.

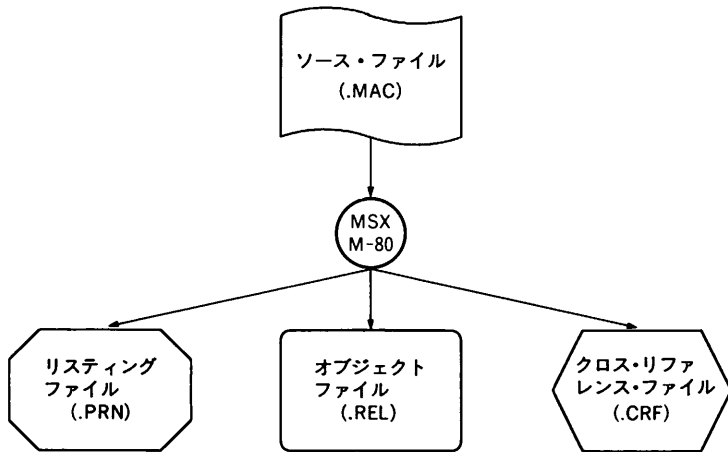



図2.9 M-80で使用するファイル

この図のうち,ソース・ファイルについてはすでにおわかりでしょう.M-80のアセンブリ言語で書かれたソース・プログラムがはいっているファイルのことです.

オブジェクト・ファイルとは,M-80がこのソース・ファイルをアセンブルして得られるオブジェクトを入れるファイルです.このファイルは次に出てくるMSX・L-80(以下L-80と呼ぶ)の入力ファイルとして用いられます.

リスティング・ファイルとは，アセンブル・リスト（1章を参照）を入れるファイルです．このファイルは，ソース・プログラムが実際にアセンブルされた結果を参照するために，プリンタに出力する場合などに用いられます．

クロス・リファレンス・ファイルは，前のリスティング・ファイルの内容に内部ラベルのクロス・リファレンス情報を含めたものです．このクロス・リファレンス情報というのは，ラベルがどの場所で定義され，どの場所から参照されているかということを示すものです．本書では扱いませんが，このクロス・リファレンス・ファイルは，CREF-80を用いるときに必要となります．

M-80を使うときには，アセンブルに必要なファイルとスイッチを指示することが必要です．このスイッチとは，M-80の持つ特別な機能を使うかどうかを指定するためのものです．スイッチについては巻末のAppendixを参照してください．アセンブルのための指示はコマンドで与えます．コマンドの内容の説明にはいる前に，コマンドの与え方を説明しておきましょう．コマンドを与える方法は次の2通りがあります．

① M-80を呼び出すときに同時にM-80のコマンドを指定する

A> M80　〈M-80のコマンド〉 ⏎

この方法では，M-80が呼び出され指定されたファイルのアセンブル作業が終わると，すぐにDOSのコマンド入力待ち状態に戻ります．この方法を用いると，M-80の起動とコマンドの指定をバッチ・ファイルとして記述できるので，アセンブル作業を一括処理することができます．

② M-80を呼び出すときにはパラメータを指定せず，M-80のプロンプト“*”に応じてコマンドを与える

A> M80 ⏎

* 〈M-80のコマンド〉 ⏎

この方法では，コマンドを終えても DOS に戻らず，次の〈M-80 のコマンド〉を入力できます．したがって，複数のファイルを一度にアセンブルしたいというような場合に有効な方法です．なお，M-80 から DOS に戻るためには M-80 のプロンプトが出ているときに CTRL ＋ C を押します．

さてこの〈M-80 のコマンド〉ですが，コマンドの形式は次のようになっています．

[〈オブジェクト・ファイル名〉][，[リスティング・ファイル名]]＝〈ソース・ファイル名〉[／〈スイッチ〉]

ここで指定するファイル名は，基本的に拡張子も含めて自由に付けることができます．しかし，図 2.9 を見てもおわかりのように，ある 1 つのプログラムについていくつかの種類のファイルが用いられるわけですから，まぎらわしい名前は避けるべきです．一般的にファイルの種類の区別は，ファイル名の拡張子の部分で行われています．M-80 では，コマンド中でファイルの拡張子が省略された場合，あらかじめ決められた拡張子が指定されたものとして処理されます．つまりその拡張子を指定したとみなされるのです．実は，サンプル・プログラムの拡張子を“.MAC”としたのは，このためだったのです．私たちが好き勝手なファイル名を付けるのは自由ですが，混乱を避けるためにも，できるだけ標準的な拡張子を付けるのが望ましいでしょう．

コマンド入力の際，オブジェクト・ファイル名とリスティング・ファイル名は省略することができます．この場合，作成されるオブジェクト・ファイルには，自動的にソース・ファイル名の拡張子を“.REL”に変えたファイル名が付けられます．したがって，ソース・ファイル名と違う名前のオブジェクト・ファイルを作りたい場合には，オブジェクト・ファイル名の指定を省略することはできません．いっぽう，リスティング・ファイル名の場合は，省略するとリスティング・ファイルが作成されません（／L スイッチを付けて強制的にリスティング・ファイルを作る場合を除いて）．

なお，次のような指定をした場合には，オブジェクト・ファイルもリスティング・ファイルも作られません．

,＝ 〈ソース・ファイル名〉

何もファイルが作られないのなら意味がないのではと思うかもしれませんが，M-80アセンブラにソース・プログラムを通すことで，エラーチェックを行うことができるのです．ただし，ここでのエラーチェックとは，アセンブラの文法に合致しているかどうかを調べるだけのもので，プログラムの内容までチェックしてくれるわけではありません．

ファイルの指定方法は複雑なので，いくつかのコマンドの例をあげておきましょう．

```
SAM1, SAM1＝SAM1
```

ソース・ファイル“SAM1.MAC”からオブジェクト・ファイル“SAM1.REL”とリスティング・ファイル“SAM1.PRN”を作成します．

```
SAM2＝SAM2
＝SAM2
```

このどちらの場合も，ソース・ファイル“SAM2.MAC”からオブジェクト・ファイル“SAM2.REL”を作成します．リスティング・ファイルは作成しません．

```
,＝SAM3
```

ソース・ファイル“SAM3.MAC”を読み込んでアセンブルを行いますが，オブジェクト・ファイルもリスティング・ファイルも出力しません．

では，実際にサンプル・プログラムをアセンブルすることにしましょう．サンプル・プログラムはワークディスク上の“KEYCODE.MAC”でした．オブジェクト・ファイル“KEYCODE.REL”とリスティング・ファイル“KEYCODE.PRN”をワーク・ディスクに出力させるには，たとえば次のように指定します．

B:KEYCODE, B:KEYCODE=B:KEYCODE

この指定によるアセンブル時の画面は図2.10のようになります.

```
A>M80 B:KEYCODE,B:KEYCODE=B:KEYCODE ⏎

No Fatal error(s)……………エラーがなければこのような
                        メッセージが表示される
A>█
```

図2.10 サンプル・プログラムのアセンブル

アセンブル時にエラーが起こらなければ"No Fatal error(s)"のようにメッセージが出力されるはずです. もしエラーがあると, その旨メッセージが表示されます(図2.11). その場合には, もう1度ソース・ファイルをエディタで修正して, 正しくアセンブルされるまでやり直してください.

```
A>M80 B:KEYCODE,B:KEYCODE=B:KEYCODE ⏎
U 0108  CD 0000            CALL    SYSTEM
U 0143  CD 0000            CALL    SYSTEM
U 015D  CD 0000            CALL    SYSTEM

3 Fatal error(s)

A>█
```

図2.11 アセンブル時のエラー・メッセージ

ここで得られたリスティング・ファイルをプリントアウトすることで，実際にアセンブルされた結果がわかります．正しくアセンブルされた場合のリスティング・ファイルは図 2.12 のように出力されます．

```
        MSX.M-8Ø 1.ØØ   Ø1-Apr-85        PAGE    1


                        ;***** PRINT KEY CODE IN HEXADECIMAL *****

                                .Z8Ø            ; We use Zilog codes

ØØØD                    CR      EQU     ØDH     ; Carrige Return
ØØØA                    LF      EQU     ØAH     ; Line Feed
ØØØ5                    SYSTEM  EQU     ØØØ5H   ; System Call Entry

ØØØØ'                           ASEG            ; Absolute SEGment
                                ORG     1ØØH    ; Set Start Address

                        ;--- Main Routine ---

Ø1ØØ    11 Ø11D                 LD      DE,MSG1 ; Print MSG1
Ø1Ø3    CD Ø141                 CALL    PUTMSG
Ø1Ø6    ØE Ø1                   LD      C,1     ; [Console Input]
Ø1Ø8    CD ØØØ5                 CALL    SYSTEM
Ø1ØB    F5                      PUSH    AF      ; Save Character
Ø1ØC    11 Ø12D                 LD      DE,MSG2 ; Print MSG2
Ø1ØF    CD Ø141                 CALL    PUTMSG
Ø112    F1                      POP     AF      ; Get Character
Ø113    CD Ø1..                 CALL    PUTHEX  ; Print code in Hex
                                        DE,MSG3 ; Print MSG3
```

図 2.12 リスティング・ファイルのプリントアウト

2-4 サンプル・プログラムのリンク・ロードと実行

アセンブルされたオブジェクト・ファイル"～.REL"を読み込んで連結(リンク・ロード)して，そのまま実行したり，実行可能なCOM形式のファイルを作る作業を行うのが，L-80リンク・ローダです．このリンク・ローダが使うファイルは図2.13のとおりです．

図2.13　L-80で使用するファイル

このようにアセンブラとリンク・ローダが分かれているのは，プログラムのモジュール別開発を行うためだということはさきほど述べました．モジュール別開発の方法についてはあとで説明することにして，ここでは，サンプル・プログラムの場合のように1つのソース・プログラムから実行可能なプログラムを作るという作業を通して，L-80の使い方に慣れておくことにしましょう．

L-80にも，M-80の場合と同様に入出力ファイル名やリンクのスイッチなどのコマンドを与えることが必要です．この指定の方法もM-80の場合とほとんど同じで，直接DOSのコマンドラインに書く方法と，L-80をパラメータを与えずに起動し，プロンプト“*”に応じてコマンドを入力していく方法の2つが用意されています．

L-80のコマンドは，基本的にはオブジェクト・ファイル名の並びという形で表されます．このオブジェクト・ファイル名は拡張子を省略することができ，その場合の拡張子は“.REL”と解釈されます．

〈オブジェクト・ファイル名1〉[，〈オブジェクト・ファイル名2〉…]

しかし，これらのコマンドを与えただけでは，各ファイルをリンクするだけで，そのリンクされたプログラムはディスクに保存されませんし，リンクが終わってもDOSに制御を戻しません．これらの働きはスイッチで指定してやらなければなりません．ここでは，必要最小限のスイッチを示しておきます．

／N　リンクした結果をこのスイッチの直前のファイル名で示されるファイルに保存する
／E　L-80からDOSに戻る

考えてみればすぐわかることかもしれませんが，／Nの直前のファイル名のデフォルトの拡張子は“.COM”となっています．ここで作られたファイル(COM形式)がDOSの外部コマンドとして実行できるのです．これ以外のスイッチで今後使うものもあるのですが，その都度説明することにします．またよく使われるスイッチについては巻末のAppendixにまとめてあります．

サンプル・プログラムのオブジェクト・ファイル“B:KEYCODE.REL”から実行可能な“B:KEYCODE.COM”を作るためのL-80に与えるコマンドは次のようになります．

```
B:KEYCODE, B:KEYCODE/N/E
```

このコマンドでリンク・ロードして“KEYCODE.COM”が完成したら，このプログラムを実際に実行してみましょう（図2.14）．

```
A>L80 B:KEYCODE,B:KEYCODE/N/E ⏎

MSX.L-80  1.00  01-Apr-85  (c) 1981,1985 Microsoft

Data    0100    0161    <   97>

42713 Bytes Free
[0000   0161          1]

A>B:KEYCODE ⏎……………さっそくできあがったコマンドを実行

Input Char ? S……………コードを出力させたい文字を入力
Hex   Code = 53H…………コードは16進数で表示される

A>█……………コマンドを実行し終わるとDOSに戻る
```

図2.14　リンク・ロードと実行のようす

以上，アセンブリ言語のプログラムを入力して実行するまでの流れをひととおり紹介してきました．確かにBASICでプログラムを書いて実行するのに比べると，アセンブリ言語によるプログラミングから実行までの手順は複雑であり，これらの手順をふむのが面倒くさいと考えるのも当然かもしれません．それにこの程度のプログラムでは，実行速度も見た目にはBASICと大差ありません．しかし，このプログラムはれっきとしたアセンブリ言語で開発されていることを思い出してください．本章では，MED，M-80，L-80というソフトウェア・ツールを使って，いつでもMSX-DOSから呼び出すことができる外部コマンド“KEYCODE”が作成できたのです．

3章

アセンブリ言語での約束事

―書式，擬似命令，マクロ機能―

BASIC言語でプログラムを書く場合にも，たとえば次のような約束事がありました．プログラムの各行の先頭に行番号を付ける，ステートメントとステートメントの間はコロン(：)で区切るなどです．同様に，アセンブリ言語のプログラムを書く場合にも，アセンブラが処理できるように書いてやらなくてはなりません．

本章では，アセンブリ言語のプログラムを書く際の約束事について説明します．プログラムの書き方といっても，ここで扱うのはプログラミングそのものではなく，アセンブラにプログラムを正しくアセンブルさせるために必要となる基本的な事項や書式についてです．また，私たちがこれから使うMSX・M-80アセンブラには，「マクロ機能」という便利な機能が備わっています．本章の後半では，このマクロ機能について簡単に説明することにします．

3 1 ソース・プログラムの書式

これからアセンブリ言語の書式や文法について見ていきますが，アセンブラと一口にいってもさまざまな種類があります．CPU が異なるとアセンブラとその文法が異なるのは当然のことです（マシン語のニーモニック自体が異なるのですから）．では，CPU が同じであればアセンブリ言語の文法はまったく同じであるか，というと残念ながらそうではありません．アセンブラによってその機能が異なりますので，基本的な部分では共通であっても，細かい部分で文法が微妙に食い違っていることがあります．

本書で扱う MSX・M-80 は，8080／Z80 の標準アセンブラである“MACRO-80”（マイクロソフト社製）と同機能なので，私たちは 8080／Z80 の標準的なアセンブリ言語の文法を取り扱うことになります．

行

前章でも述べましたが，ソース・プログラムは「行」の集まりとして扱われます．そして各行に，シンボル，ラベル，CPU 命令(ニーモニック)，擬似命令，コメントなどが置かれています．ここでの行は，画面やリストの左端からリターン・キーで改行されるまでの文字列を指しています．画面やプリンタ上で複数行にまたがって表示されたとしても，1 行として取り扱われるという点は BASIC などと変わりありません（図 3.1）．

また，リストではとくにリターンコードを表示していませんが，入力する際には忘れないようにしてください．

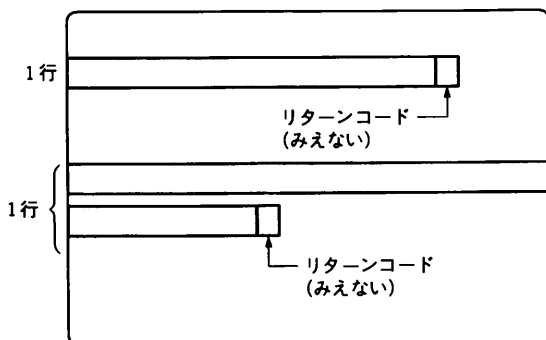



図 3.1　行の概念

ステートメントとフィールド

プログラムの各行には，ステートメント（Statement＝文）が置かれます．このステートメントの形式は図 3.2 のようになっています．

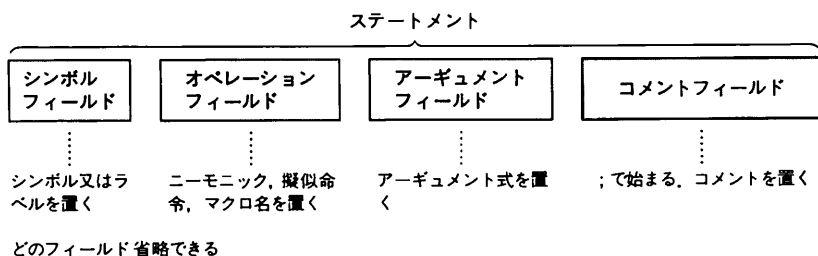



図 3.2　ステートメントの形式

・シンボル・フィールド

このフィールドにはシンボルを記述します．ラベルの場合には，その直後にコロン（：）を付けなくてはなりません．

・オペレーション・フィールドとアーギュメント・フィールド

このフィールドには，CPU 命令，擬似命令などを記述します．

・コメント・フィールド

セミコロン（;）を先頭にして，それ以降にコメントを記述します．

また，それぞれのフィールドの間は，1つ以上のスペースまたはTABコードで区切らなくてはなりません．TABコードは[TAB]を押して入力されるコードです（図3.3）．

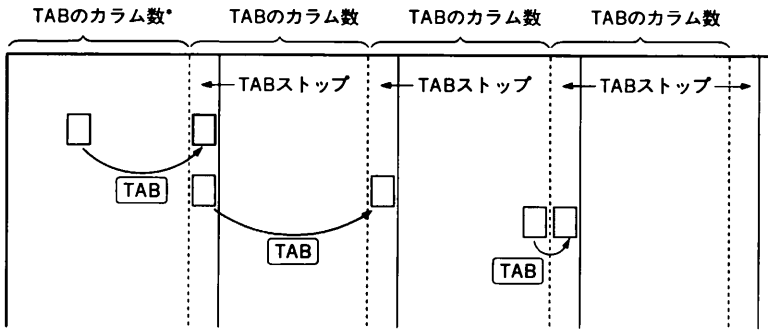



図3.3 TABコードの働き

このようにTABコードを利用することで，簡単にフィールドの桁を揃えることができます．また，どのフィールドも省略することが可能となっています．さらに，まったくステートメントのない空行も許されています．

なお，MSX・M-80はカタカナや漢字を正しく解釈しないため，コメントも英語かローマ字で記述する必要があります．しかし，本書ではプログラムの内容をより深く理解してもらうため，これ以降に出てくるプログラムについては，リストの横に日本語の説明を入れてあります．コメントはできる限り付けておいた方が，あとでプログラムを見直すときにわかりやすいものです．そこで，みなさんがプログラムを打ち込む際には，本書の説明を参考にして，自分でコメントを付けておくとよいでしょう．

シンボル

シンボルは，数値やアドレスに付ける名前です．このシンボルについては文字数や使える文字などに制限（アセンブラによって異なる）があります．MSX・M-80では，任意の長さのシンボルを用いることができますが，先頭の16文字までが有効です．シンボルに使える文字は次のものです．

A～Z　a～z　0～9　$　.　?　@　_

このうち，数字（0～9）は先頭の文字に用いることができません．これは，数値との混同を避けるためです．また，英小文字と英大文字は区別されません．

また，シンボルによって表現可能な数値の範囲は0～FFFFH，10進数で0～65535です．

シンボルの定義は，コロン（：）を用いる方法と，擬似命令EQUを用いる方法が一般的です．いったん定義されたシンボルは，CPU命令や擬似命令のアーギュメントとして参照することができます．

数値定数・文字定数

これらの定数はいずれもCPU命令や擬似命令のアーギュメントとして用いられます．数値定数の基数（何進数であるか）を示すためには，次の記号を用いますが，これらを省略した場合には10進数だと解釈されます．16進数の場合に最上位にA～Fがくるときには，シンボルと区別するためにその前に0を付けなくてはなりません．

nnnnB	……2進数
nnnnO or nnnnQ	……8進数
nnnnD	……10進数
nnnnH or X'nnnn'	……16進数

また，'A'のような文字定数を用いることもできます．'A'の値は対応する文字コードですから41Hです．たとえば，

```
CP      'A'
```

と書くのは，

```
CP      41H
```

とするのと同じ意味を持つのです．

文字列

文字列はシングル・クォーテーション(')，またはダブル・クォーテーション(")で囲みます．文字列がアーギュメントとなると引用符内の文字列が順にメモリに格納されます(擬似命令DBの解説(p.69)参照)．

演算子

数値定数，文字定数，シンボルなどは演算を行わせて，その演算結果をCPU命令や擬似命令のアーギュメントとして与えることができます．この演算子は，算術演算子と論理演算子の2つに分類されますが，このうち真偽の判断のための論理演算子についてはここでは触れません．

算術演算子は，四則演算や，剰余(割り算の余り)，ビットのシフト，ビット単位の論理演算などがあげられます．このうち，四則演算以外の演算子，つまり，−，+，*，／以外の演算子は，その演算の対象となる定数やシンボルなどと分離するために1つ以上の空白を入れなければなりません．

また，演算子には優先順位が設定されています．たとえば，かけ算の演算子（*）は，足し算の演算子（+）よりも優先順位が高いといった具合いです．この演算子（算術演算子）と優先順位を表3.1に示し，演算の具体的な例を図3.4に示します．

演算子	演　算　内　容	優先順位
HIGH	絶対16ビット値の上位8ビットを分離する	1
LOW	絶対16ビット値の下位8ビットを分離する	
＊	右オペランドと左オペランドを乗ずる	2
／	左オペランドを右オペランドで除し，商を返す	
MOD	左オペランドを右オペランドで除し，剰余を返す	
SHR	左オペランドを右オペランドの値だけ右へシフトする	
SHL	左オペランドを右オペランドの値だけ左へシフトする	
−(負号)	オペランドの負数を返す	3
＋	左オペランドと右オペランドを加算する	4
−	左オペランドから右オペランドを減ずる	
NOT	オペランドの全ビットを反転する.	5
AND	左オペランドと右オペランドの論理積を返す	6
OR	左オペランドと右オペランドの論理和を返す	7
XOR	左オペランドと右オペランドの排他的論理和を返す	

表 3.1　演算子と優先順位

図 3.4　演算の例

擬似命令

1章でも述べたように，擬似命令はアセンブラに対して指示を与える命令です．ですから，擬似命令がアセンブラの機能を表すといってもよいでしょう．MSX･M-80は非常に高機能なアセンブラなので擬似命令も多く用意されています．MSX･M-80の擬似命令は大きく次の7種類に分けられます．

・ロケーション・モード関連 …… セグメントの割り当て
・文の制御 …… ニーモニック記法やプログラムの終わりなどの指定
・データ定義／シンボル定義 …… データ領域の確保や値の設定，シンボルの定義
・ファイル操作 …… ファイルの読み込みなど
・リスト出力制御 …… アセンブル･リストのタイトルや書式を制御する
・条件判断 …… 条件によりアセンブルの流れを変える
・マクロ関連 …… マクロの定義など

いろいろな種類の命令があることに驚かれた人もいるでしょうが，結局これらの命令はアセンブラによるプログラム開発を助けるためのものですから，必要なものから順にマスターしていけばよいのです．ここでは，この多くの擬似命令のなかでもアセンブラ･プログラミングに欠かせないもの，とくに有用なものについてのみ説明することにします．その他の命令については，一部はあとで扱うことにしますが，それ以外の命令は巻末のAppendixやMSX･M-80のマニュアルなどを参照してください．

ロケーション・モード指定

● ASEG　(Absolute SEGment)　……　**絶対モード指定**

この ASEG 擬似命令は，生成されるプログラムを置くアドレスをアセンブル時に決めて固定するという指定です．「なんだ当り前のことではないか」と思われるかもしれません．

MSX・M-80 では何も指定しない場合には，プログラムを置くべきアドレスをアセンブル時には決定しません．つまり，アセンブルして得られた結果(この段階では実行することはできない)は，任意のアドレスに置くことができるのです．このことを，「**リロケータブル**」であるといいます．このアセンブル結果は，リンク・ロードという作業をとおして実際に置くべきアドレスを決め，メモリ上に配置するのです（図 3.5）．

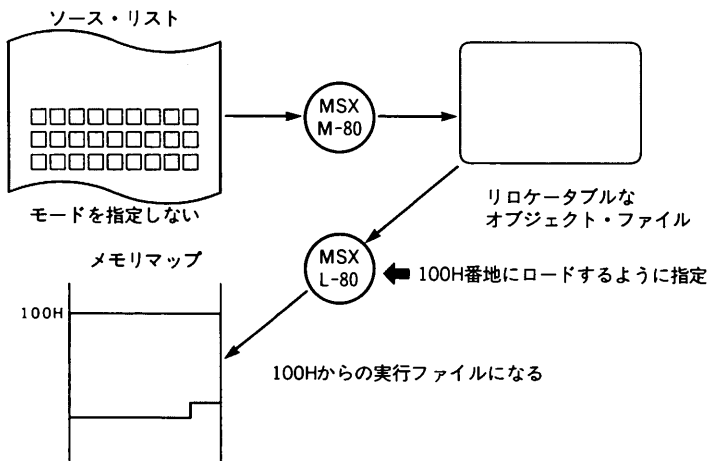



図 3.5　モードを指定しない場合（相対モード）

これに対して，絶対モード指定をしてアセンブルしたプログラムは，アセンブル時に定められたアドレスにしかリンク・ロードできません（図 3.6）．

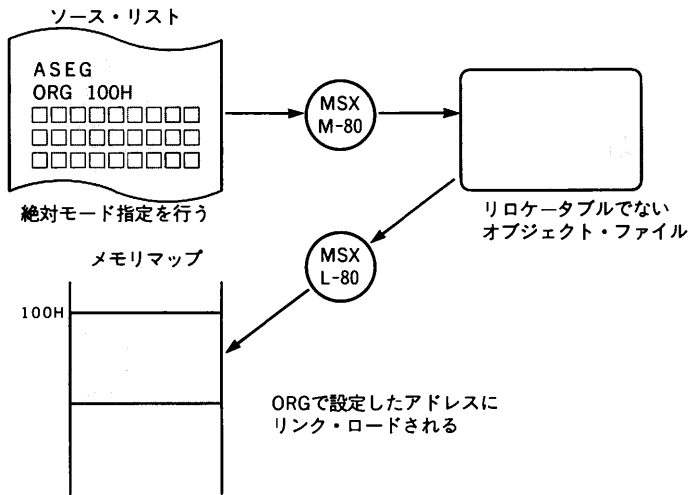



図 3.6 絶対モード指定をした場合

実は,「アセンブルしたあとにロード・アドレスを決められる」という機能は, MSX・M-80 の持つ特徴の 1 つなのです. この機能の使い方などについては 4 章で詳しく説明します.

文の制御

● ORG (ORiGin) …… ロケーション・カウンタ値の設定

ロケーション・カウンタの値を設定するものです. ロケーション・カウンタとは, 得られるオブジェクト・プログラム (マシン語コード) のアドレスを指しています. 結局この命令でオブジェクト・プログラムをメモリ上に置く際のアドレスを設定することになります. 絶対モードでは, "ORG XXXXH" と書かれていると, この行以降のオブジェクト・プログラムをアドレス XXXXH 番地から生成していきます. 1 つのソース・プログラムのなかで複数の ORG 擬似命令が存在することも許されています. しかし, この場合には, プログラムが重なってしまうことも起こり得ますから注意が必要です(図 3.7).

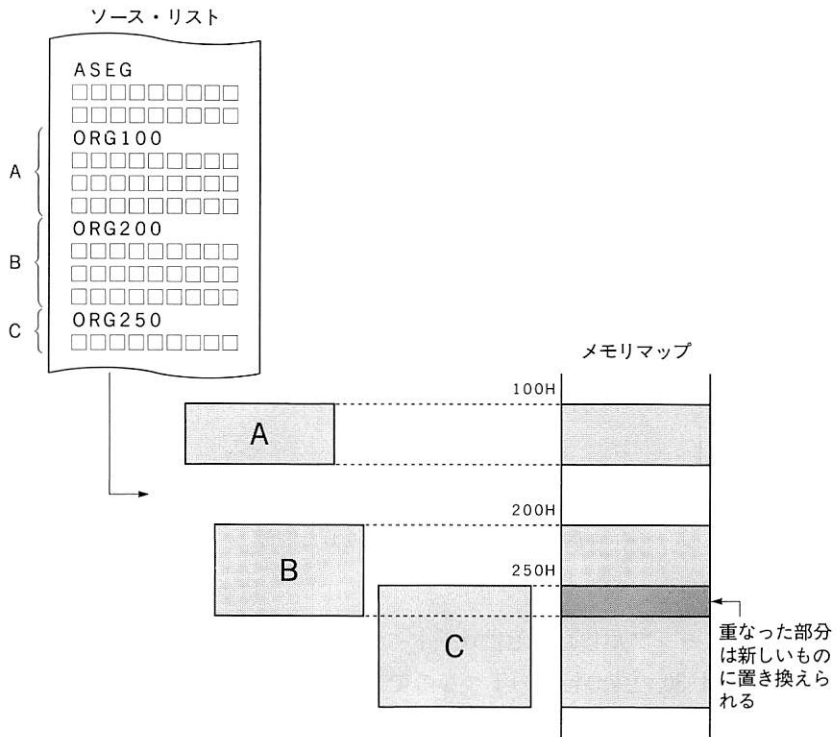



図 3.7　複数の ORG 命令のあるプログラム

●.Z80　……　Z80 モード選択

●.8080　……　8080 モード選択

MSX・M-80 は，Z80 CPU の命令セットおよび 8080CPU の命令セットのどちらでもアセンブルすることができます．この命令セットの選択は，この擬似命令を用いてプログラム中で行うか，アセンブルを行わせるときにコマンドラインからスイッチを用いて指定します．

● END　……　アセンブルの終了

ソース・プログラムの終了を指定します．END 命令以降に何が書かれてあっても，その部分はアセンブルされません．

シンボルの定義

● EQU (EQUate) …… 再定義不可能なシンボルの定義

シンボルに数値を割り当てます．この書式は次のようになっています．

〈シンボル〉 EQU 〈式〉

このとき，〈シンボル〉に〈式〉の値が定義されます．〈式〉には，数値定数のほかにもそれまでに定義されたシンボルや，それらの間の演算結果を用いることができます．ラベルの定義ではありませんから，〈シンボル〉のあとには，コロン（：）は付けません．なお，いったん擬似命令 EQU で定義されたシンボルは，そのプログラム中で再定義することはできません．

● ASET または SET …… 再定義可能なシンボル定義

この擬似命令 ASET／SET は，擬似命令 EQU と同様にシンボルに値を定義するものです．その書式は，EQU の場合と同じです．擬似命令 EQU と異なっているのは，プログラム中で何度でも再定義できるという点です．なお Z80 のニーモニックには SET 命令がありますから，擬似命令の SET は Z80 モードでは用いることができません．Z80 モードでは，機能はまったく同等の擬似命令 ASET を用いてください．

データの定義

● DB (Define data Byte) …… バイト・データの定義

メモリ上にデータを1バイト単位で格納するための命令です．その書式は次のようになっています．

```
DB   〈式〉[，〈式〉……]
DB   〈文字列〉[，〈文字列〉……]
```

ここでは，〈式〉の場合と〈文字列〉の場合に分けて書きましたが，実際には2つの組み合わせもよく用いられます．なお，〈式〉の値は2バイトで計算されて下位1バイトデータがセットされますが，〈式〉の値の範囲は−255〜＋255の範囲でなければなりません(それ以外のときにはエラーとなる)．また，〈文字列〉は1バイトごとに最上位ビットを0にして順に格納されます．

この擬似命令DBの使用例を図3.8に示します．

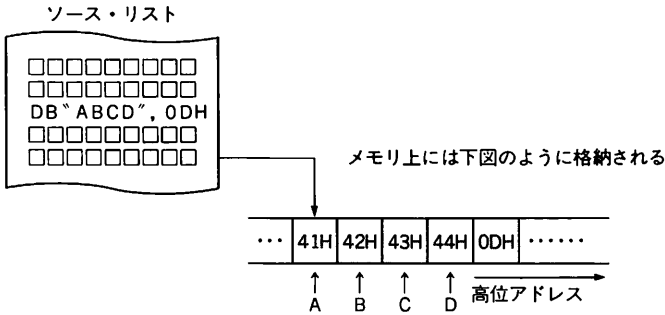



図3.8 擬似命令DBでのデータの格納

● DW (Define data Word) …… 2バイト (ワード)・データの定義

メモリ上にデータを2バイト単位で格納するための命令で，その書式は次のとおりです．

```
DW 〈式〉[,〈式〉……]
```

〈式〉の値は，基本的に0〜65535，つまり16ビット符号なしの整数ですが，−32768〜−1の場合には，2の補数表記として扱われます．この2の補数について詳しくは説明しませんが，FFFFHを−1，FFFEHを−2というように解釈するというものです．

いずれにせよ，この値がメモリ上では，下位バイト，上位バイトの順で格納されます (図3.9)．

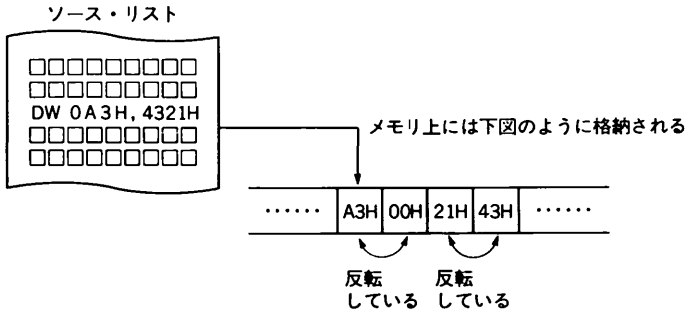



図 3.9 擬似命令 DW でのデータの格納

● DS (Define Storage) …… メモリ領域の確保と初期設定

メモリ領域を確保（予約）します．この擬似命令には次の 2 つの書式があります．

① DS 〈式〉
② DS 〈式〉，〈値〉

①の場合，メモリ領域を〈式〉の値が示すバイト数だけ確保します．つまり，ロケーション・カウンタを〈式〉の値だけ進めるのです．この場合には，確保されたメモリ領域のプログラム実行開始時の初期値は決っていません．この初期値が決まらない領域をどう使うかがわからないかもしれませんが，たとえばプログラムを実行するときに領域の内容を初期設定するような場合や，データの一時退避用の領域などに用いればよいのです．

②の場合にも，①の場合と同様にメモリ領域を〈式〉の値が示すバイト数だけ確保します．①と違っているのは，その初期値が〈値〉に設定されるという点です．

図 3.10 にこの命令を用いた領域の確保のようすを示します．

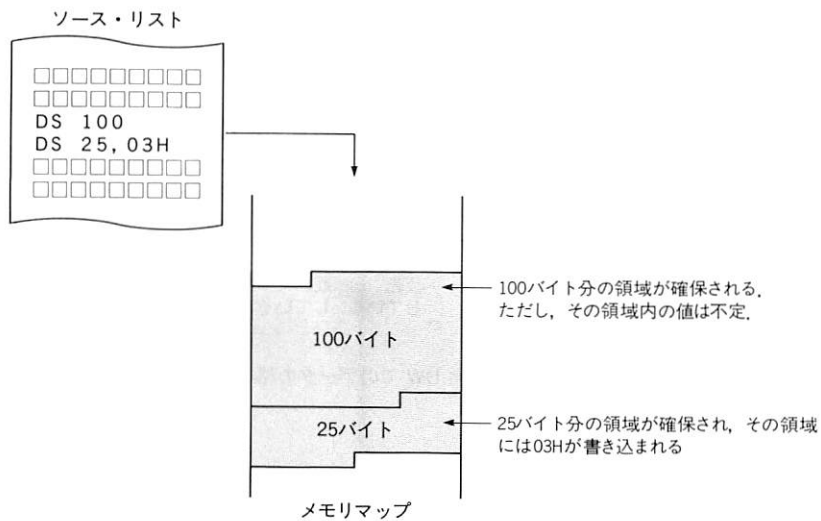



図 3.10　擬似命令 DS でのメモリ領域の確保

3 3 マクロ機能

マクロ機能とはプログラムのなかで一連の命令群をある名前で登録しておき，必要なところにその名前で命令群を呼び出せるというものです．シンボルがアセンブル時に数値に変換してくれるということに少し似ています．その程度の機能ではたいしたことではないと思う人もいるかもしれませんが，このマクロ機能を使うと，アセンブリ言語のプログラムがより一層わかりやすく簡潔に書けるようになるのです．

マクロの定義

マクロは文字列の置き換えですが，マクロを利用するためにまずこのマクロを登録（マクロ定義）しておかなくてはなりません．このマクロの定義は，ソース・プログラム中で次のように行います．

```
〈マクロ名〉  MACRO      マクロ定義の開始
               •
               •        マクロ定義ブロック
               •
              ENDM      マクロ定義の終了
```

こうすると，マクロ定義ブロックの内容が〈マクロ名〉で登録されます．この〈マクロ名〉は，シンボルの場合と同様に英文字で始まる文字列ですが，頭の16文字までが有効です．もちろん，マクロ定義の内容は何行にわたってもよく，ニーモニック，コメント，擬似命令，マクロ呼び出しなどアセンブラで処理できるものなら何でも書くことができます．

たとえば画面上で改行を行うというマクロは，1文字表示のシステムコー

ル〈02H〉を利用して次のように書くことができます.

```
CRLF    MACRO      …… CRLFというマクロの定義の開始
LD      E,0DH      …… キャリッジ・リターン・コード
LD      C,02H      …… 1文字出力のファンクション番号
CALL    0005H      …… システムコールを呼ぶ
LD      E,0AH      …… ライン・フィード・コード
LD      C,02H      …… 1文字出力のファンクション番号
CALL    0005H      …… システムコールを呼ぶ
ENDM               …… CRLFマクロの定義の終わり.この部分
                      は定義するだけで実際にマシン語として
                      は展開されない
```

マクロの呼び出し

マクロの呼び出しは，マクロの定義以降ならどこでも可能です．マクロの呼び出す手順は簡単です．CRLFというマクロ名で定義されたマクロを呼び出すには，必要な場所に，

```
CRLF
```

と書くだけです．こうすると，その場所にマクロ定義ブロックの内容が復元されるのです．このマクロの呼び出しでマクロの内容が復元することを「**マクロを展開する**」といいます．

このようにあらかじめマクロを定義しておけば，あたかもアセンブリ言語に新しい命令が増えたかのように使えるために，プログラムを読みやすく書きやすいものにすることができるのです．

さきほどの改行を行うマクロを実際に定義し，呼び出すプログラム"CRLF.MAC"のアセンブル・リストを見てみましょう（図3.11）.

```
                        ; PUT CR/LF USE MACRO

                                .Z80

                        CRLF    MACRO
                                LD      E,0DH
                                LD      C,2
                                CALL    0005H     CRLFマクロの定義
                                LD      E,0AH
                                LD      C,2
                                CALL    0005H
                                ENDM

  0000'                         ASEG
                                ORG     100H

                                CRLF
  0100    1E 0D         +       LD      E,0DH
  0102    0E 02         +       LD      C,2
  0104    CD 0005       +       CALL    0005H  ·······CRLFマクロ
  0107    1E 0A         +       LD      E,0AH         が展開されて
  0109    0E 02         +       LD      C,2           いる
  010B    CD 0005       +       CALL    0005H
  010E    C9                    RET

                                END
                        ·········マクロが展開されている
Macros:                          部分であることを示す
CRLF ··········定義されたマクロがリスト・アップされる

Symbols:
```

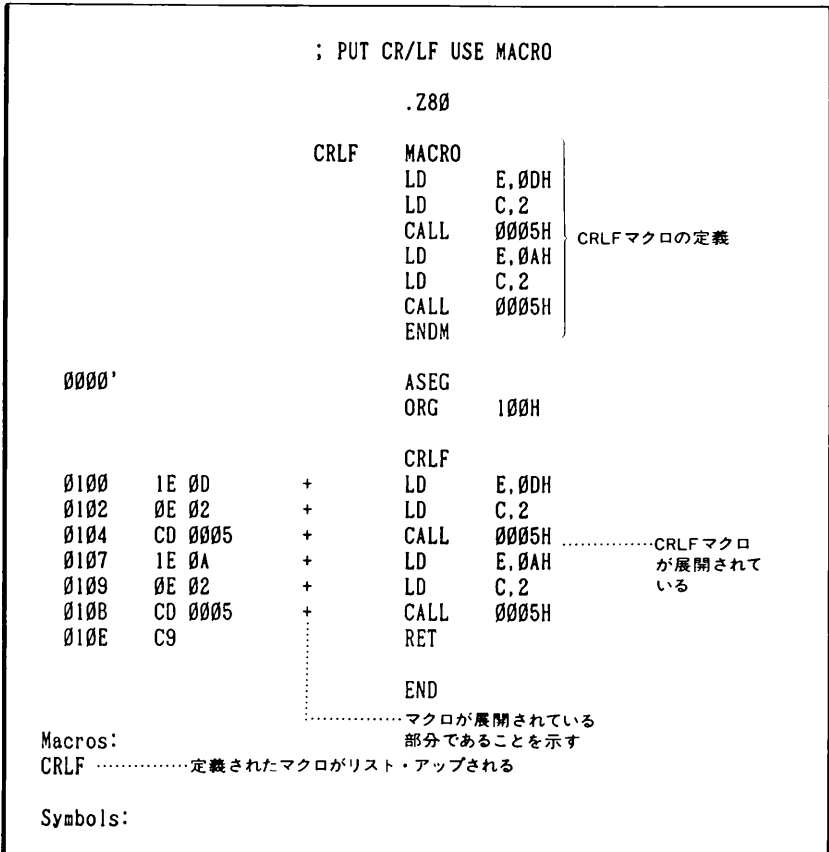
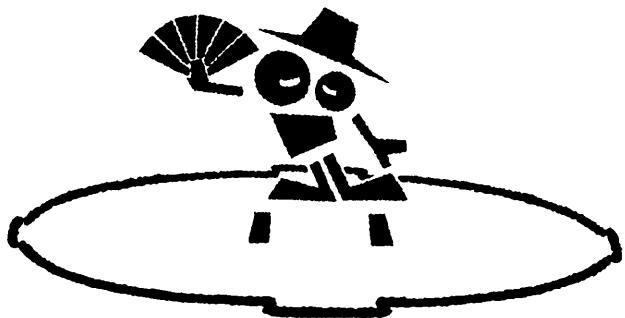



図 3.11　CRLF.MAC のアセンブル・リスト

パラメータ付きマクロ

これまでの例は単なる文字列の置き換えでしたが，基本パターンが同じでもパラメータに従ってマクロの展開を変化させるということもできます．こうすれば，1つのマクロ定義をいろいろな場面で使用することができます．このマクロ定義の書式は次のようになります．

```
〈マクロ名〉  MACRO  [〈ダミー〉[,〈ダミー〉[,  ……  ]]]
                •
                •
                •
              ENDM
```

この書式は，パラメータを取らない場合のマクロ定義の書式を拡張したものです．ここで〈ダミー〉を「**仮パラメータ**」（ダミー・パラメータ）と呼びます．仮パラメータはそれぞれ32文字以内です．この仮パラメータを使ってマクロ定義ブロックを記述することができます．

このマクロを呼び出すためには，マクロ定義をしたあとで

```
〈マクロ名〉  [〈パラメータ〉[,〈パラメータ〉[,  ……  ]]]
```

と書いておきます．そうすると，〈パラメータ〉の内容を“左から順に”マクロ定義の〈ダミー〉に置き換えて，その場に展開してくれるのです．マクロを呼び出すときに与える〈パラメータ〉を「**実パラメータ**」といいます．なお，マクロ定義のときの仮パラメータの数とマクロ呼び出しのときの実パラメータの数はかならずしも一致している必要はありません．余っているパラメータは無視しますし，不足しているパラメータは空白として扱われるからです．

では，実例を示すことにしましょう．図3.12は画面上に文字列を表示するプログラム(PUTMSG.MAC)のアセンブル・リストです．パラメータがどの

ように展開されているかを見てください。

```
                              ; PUT MESSAGE ON SCREEN

                                      .Z80

0024                          EOS     EQU     '$'
0005                          SYSTEM  EQU     0005H

                              PUTCHR  MACRO   CHAR     ┐
                                      LD      E,CHAR   │ PUTCHRマ
                                      LD      C,02H    │ クロの定義
                                      CALL    SYSTEM   │
                                      ENDM             ┘

                              CRLF    MACRO            ┐ CRLFマクロの
                                      PUTCHR  0DH      │ 定義(PUTCHR
                                      PUTCHR  0AH      │ マクロを用いて
                                      ENDM             ┘ いる)

                              PUTMSG  MACRO   MSGP     ┐
                                      LD      DE,MSGP  │ PUTMSGマ
                                      LD      C,09H    │ クロの定義
                                      CALL    SYSTEM   │
                                      ENDM             ┘

0000'                                 ASEG
                                      ORG     100H

                                      CRLF
0100    1E 0D          +              LD      E,0DH  ←── パラメータが展開
0102    0E 02          +              LD      C,02H       されている
0104    CD 0005        +              CALL    SYSTEM
0107    1E 0A          +              LD      E,0AH  ←
0109    0E 02          +              LD      C,02H
010B    CD 0005        +              CALL    SYSTEM ---
                                      PUTMSG  MSG
010E    11 0125        +              LD      DE,MSG ←
0111    0E 09          +              LD      C,09H
0113    CD 0005        +              CALL    SYSTEM ---
                                      CRLF
0116    1E 0D          +              LD      E,0DH  ←
0118    0E 02          +              LD      C,02H
011A    CD 0005        +              CALL    SYSTEM
011D    1E 0A          +              LD      E,0AH  ←
011F    0E 02          +              LD      C,02H
0121    CD 0005        +              CALL    SYSTEM ------マクロが展開されている
```

```
  0124    C9                     RET

  0125    44 6F 6E       MSG:    DB      "Don't Worry My Friend!",EOS
  0128    27 74 20
  012B    57 6F 72
  012E    72 79 20
  0131    4D 79 20
  0134    46 72 69
  0137    65 6E 64
  013A    21 24


                                 END

Macros:
CRLF            PUTCHR           PUTMSG ··············定義されたマクロがリスト
                                                      アップされる
Symbols:
0024    EOS             0125     MSG               0005    SYSTEM
```

図 3.12　PUTMSG.MAC のアセンブル・リスト

この例では，マクロ CRLF のなかで別のマクロ PUTCHR を呼び出しています．このようにマクロのなかで別のマクロの展開を指示することもできます．

以上のように，まとまった機能ごとにマクロを定義しておけば，あとは名前(マクロ名)だけで，それらの機能が簡単に呼び出せるということがわかってもらえたと思います．

マクロとサブルーチン

マクロはサブルーチンと比較してどのように違うのでしょうか？　どちらも名前で命令の集まりを「呼び出す」というところは似ています．しかし，マクロは呼び出すといっても命令の集まりを“CALL”するのではなく，そのつど命令をソース・プログラムの呼び出す部分に展開しています(図 3.13)．

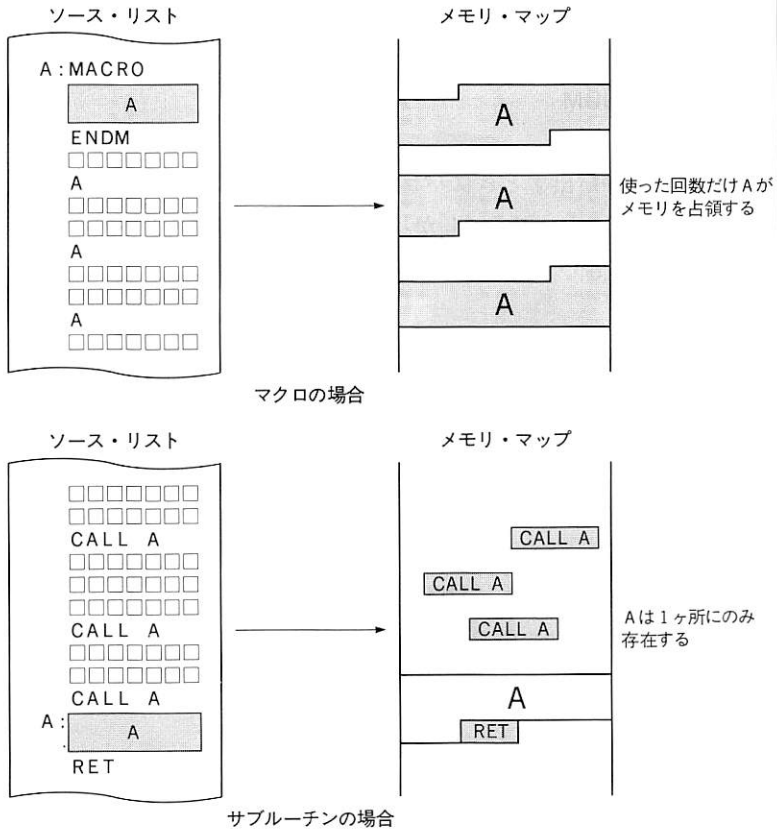



図 3.13 マクロとサブルーチン

したがって，マクロを多用すると使った分だけ命令が展開されるため，多くのメモリ領域を必要とします．しかし CALL～RET 命令が使われないためにプログラムの実行が少し速くなるというメリットもあります．このあたりを考えてマクロとサブルーチンを使い分ける必要があります．

また，マクロを展開するのはあくまでアセンブルを行うときであって，実行するときではありません．「A レジスタにパラメータの 2 倍を加える」というマクロを次のように書いたとします．

```
ADDA2   MACRO   PARAM
        ADD     A,PARAM*2
        ENDM
```

このマクロを呼び出すときにパラメータとして定数を渡すのであれば，まったく問題は起こりません．しかし，たとえばBレジスタの内容を渡そうとして次のように書いても正しくマクロ展開されません．

```
ADDA2   B
```

どうしてもレジスタ値を与えたいのならば，次のようにマクロを定義する必要があります．

```
ADDA2   MACRO   PARAM
        ADD     A,PARAM
        ADD     A,PARAM
        ENDM
```

繰り返し

M-80には，特殊なマクロとして回数を指定すれば回数分だけ命令を展開してくれるという便利な擬似命令が用意されています．その書式は次のとおりです．

```
REPT    <回数>
  ・
  ・
  ・
ENDM
```

こうすることで，このREPTとENDMの間の部分を〈回数〉だけ展開してくれるのです．当然のことですが，この命令の展開はアセンブル時に行われるものですから，〈回数〉はアセンブル時に決まっている必要があります．

では，これまでのマクロ機能のまとめとして，1章で取り上げた，入力キーの文字コードを表示するプログラムをマクロ機能を使って書き直してみます．

```
;***** PRINT KEY CODE IN HEXADECIMAL *****……1章のサンプル・プログラムとの
                                            違いはマクロを使用している点
        .Z80                                だけでプログラムの構造はまった
                                            く同じ

CR      EQU     0DH
LF      EQU     0AH
SYSTEM  EQU     0005H

SYSCALL MACRO   FUNCNO……………システムコールをするマクロの
        LD      C,FUNCNO       定義．パラメータとしてファンク
        CALL    SYSTEM         ション番号をとる
        ENDM
                            メッセージを表示するマクロの定義(1章の
                            サンプル・プログラムではサブルーチンとな
PUTMSG  MACRO   MSG………… っていた)．パラメータは表示するメッセー
        LD      DE,MSG      ジのはいったアドレス
        SYSCALL 09H……………システムコール・マクロの呼び出し
        ENDM

        ASEG
        ORG     100H

;--- Main Routine ---

        PUTMSG  MSG1……………文字列表示マクロ(MSG1の表示)
        SYSCALL 01H……………システムコール・マクロ(1文字入力)
        PUSH    AF
        PUTMSG  MSG2……………文字列表示マクロ(MSG2の表示)
        POP     AF
        CALL    PUTHEX
        PUTMSG  MSG3……………文字列表示マクロ(MSG3を表示)
        RET

MSG1:   DB      CR,LF,'Input Char ? $'
MSG2:   DB      CR,LF,'Hex   Code = $'
MSG3:   DB      'H',CR,LF,'$'
```

```
;--- Print Acc in Hexadecimal Form ---··············Aレジスタの内容
PUTHEX:                                           を16進数表示する
                                                  ルーチン
        PUSH    AF
        REPT    4 ··············リピート・マクロ(4回繰り返して展開される)
        RRCA··············繰り返される内容
        ENDM··············リピート・マクロの終わり
        CALL    HEXSUB
        POP     AF
HEXSUB:
        AND     ØFH
        ADD     A,'Ø'
        CP      '9'+1
        JR      C,PUTH2
        ADD     A,'A'-1Ø-'Ø'
PUTH2:
        LD      E,A
        SYSCALL Ø2H ··············システムコール・マクロ(1文字表示)
        RET

        END
```

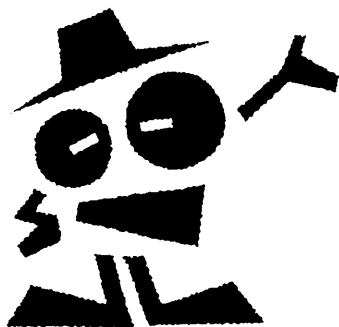
リスト 3.1　KEYCODE2.MAC

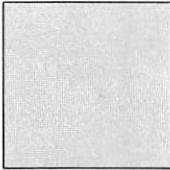
とりあえずマクロ機能がどのようなものかはなんとなくわかってもらえたと思います．マクロ機能は確かに便利なものですが，ないと何もできなくなるという種類のものではありません．プログラムのなかで少しずつ使っていくようにするとより理解も深まっていくことでしょう．

本書ではこれ以上マクロ機能について説明しませんが，マクロ機能をアセンブル時の条件判断機能などと組み合わせて使うともっと複雑な処理を行わせることもできます．もっと詳しく知りたい人は，マニュアルや他の書籍を参照してください．

4章

プログラム開発の効率化

―プログラムをモジュール別に開発する―

私たちが使っているM-80は，多くの優れた特長を持っています．その1つは3章で説明したマクロ機能です．そしてもう1つはアセンブルして直接マシン語プログラムを出力するのではなく，いったんオブジェクト・ファイルを出力するということです．2章でも少し触れましたが，このオブジェクト・ファイルを使うことでプログラムをモジュール別に開発できるようになります．このプログラムのモジュール別開発は，より実用的なプログラムを作るためにはどうしても必要となってきます．

本章では，M-80のオブジェクト・ファイル生成という機能を中心にモジュール別開発について考えていきます．また，本章の最後では，この手法を使って時計のプログラムを作ってみることにします．

4 1 モジュール化の意義

2章で作成したプログラムは1つのソース・ファイルにすべての処理を記述していました．このようにプログラム自体が短いものであれば，プログラムをいくつかにわける必要性はあまり感じないでしょう．しかし，プログラムの規模が大きくなりソース・プログラムの量が多くなった場合でも，このままでよいのでしょうか？

アセンブリ言語では，違う値に同じ名前のシンボルを定義することはできません．ですから，新しいルーチンを作る際には，そのなかで使うすべてのシンボルがまだ使われていないことを確かめる作業が必要になりますが，この作業はプログラムが長くなるほど大変なものとなります．

また，長いプログラムを書いていると，構造のはっきりしない，いわゆるスパゲッティ・プログラムになりかねません（図4.1）．

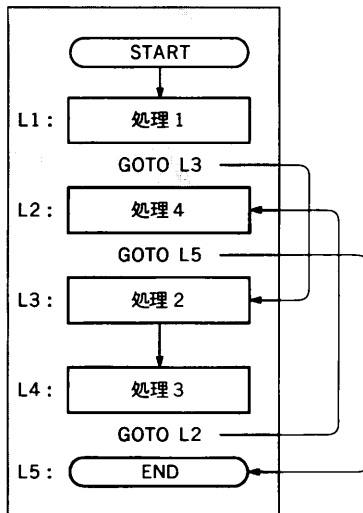



図4.1　長ったらしいスパゲッティ・プログラム

このような事態を避けるためには，まずプログラムをサブルーチン単位に分けることが必要です．さらに，機能の似通ったサブルーチンを集めてモジュール化することで，より見通しのよいプログラムにすることができます（図 4.2）．

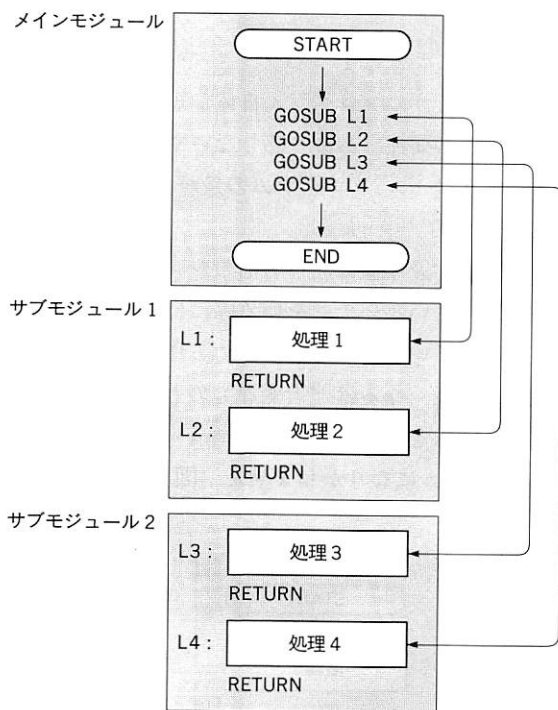



図 4.2　モジュール分割

このモジュール別のプログラム開発の利点をまとめると次のようになります．

・プログラム全体をモジュールの集まりとして扱うので，階層的な管理ができる
・各モジュール単位で開発できるので，他のモジュールの変更の影響が少なく，結果として開発効率がよくなる
・デバッグ作業では，修正する必要があるモジュールだけをチェックすればよい
・モジュールごとに機能を分けるようにすれば，プログラムが構造的に書けるようになる

次に，M-80を使ってアセンブリ言語のプログラムをモジュールごとに分けて開発する作業の手順を**図4.3**に示します．この図と，1つのソース・プログラムで開発をする場合（2章の図2.1）とを比較してみてください．

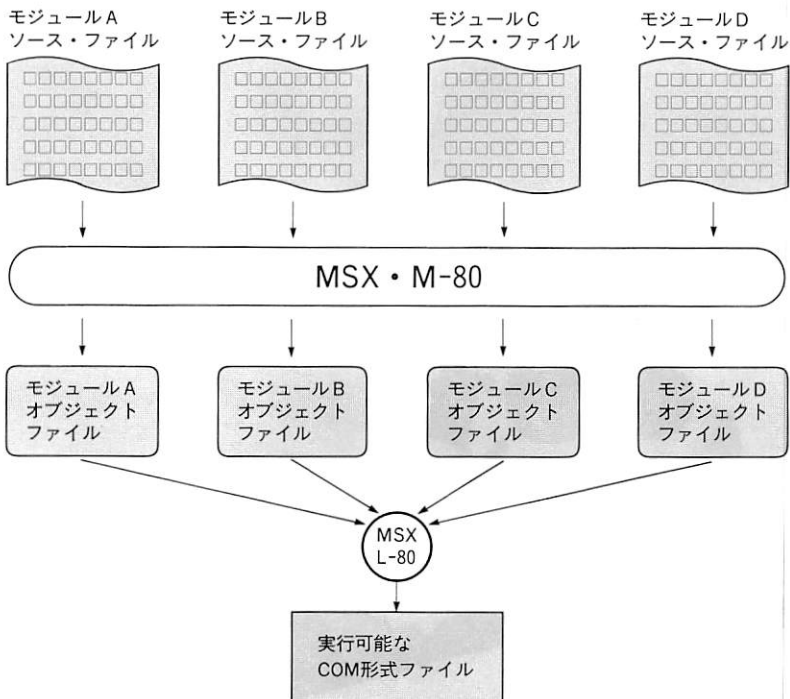
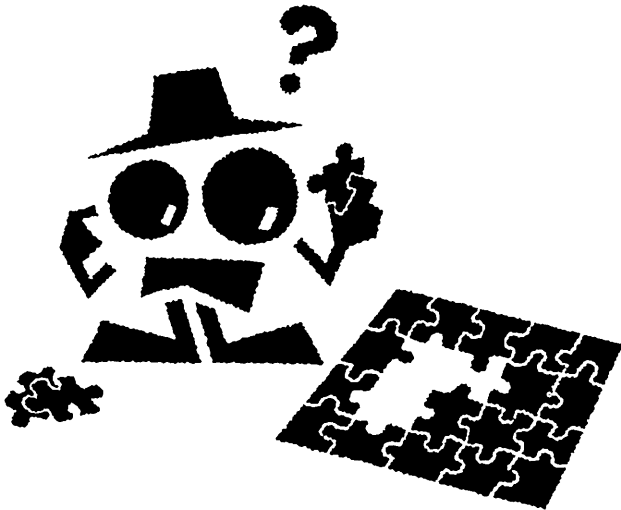



図4.3 モジュール別のプログラム開発

ここで注意しておきたいことが1つあります．アセンブル時には，それぞれのモジュールの大きさは互いに知ることはできません．ですから，先頭に置かれるモジュールはロードされるアドレスを指定するので決めることができるとしても，それに続く残りのモジュールの置かれるアドレスは，先頭のモジュールの大きさがわかるまで知ることができないのです．

このように，それぞれのモジュールがアセンブルされているときにはプログラムの置かれる場所（絶対アドレス）は決まっていません．そのため，アセンブルして得られるオブジェクト・ファイル“~.REL”は，モジュール内のラベルをそれぞれのモジュールの先頭からの相対アドレスの形で持っているのです．L-80リンク・ローダがそれぞれのモジュールをロードするアドレスを決めて，リロケータブル形式のモジュールを接続（リンク・ロード）することになるのです．

4-2 モジュール化とシンボル

前にも述べましたように，1つのモジュール内では違う場所に同じ名前のラベルを付けること，つまり1つのシンボルにいくつかの値を持たせることはできません．この点はモジュール分割せずに，1つのソース・プログラムでプログラム全体を記述する場合と同様です．しかし，それぞれのモジュールは独立して扱われるので，異なるモジュールであれば同じシンボル名を使っていても，それらは区別して扱われます．このため，他のモジュールで同じシンボル名を使ったかどうかをいちいちチェックする必要はなくなります．このようなシンボル，つまりそれぞれのモジュール内だけで使われるものを**「内部シンボル」**といいます．この内部シンボルは，アセンブル時にその内容に置き換えられます．

いっぽう，モジュール間で互いのシンボルを共同で使うことがあります．モジュール間で使われるシンボルは**「外部シンボル」**と呼ばれます．この外部シンボルは，たとえばあるモジュールで定義したサブルーチンを他のモジュールから呼び出すときなどに使われます．そのサブルーチンのシンボルは呼び出す側のモジュールでは定義していないのですから，そのままアセンブルするとエラーになります．こうならないようにするためには，内部シンボルと外部シンボルとの区別をはっきりさせる必要があります．M-80では，あるシンボルがモジュール間にまたがって使われることを宣言する疑似命令“PUBLIC”（パブリック）と“EXTRN”（エクスターナル）の2つが用意されています．これらの使い方の例を示します．

PUBLIC ABC ……………	ABCというシンボルを他のモジュールから参照できるようにする
EXTRN ABC …………	ABCというシンボルは他のモジュールでPUBLIC宣言して定義されている

このPUBLICとEXTRNの考え方を示すと図4.4のようになります.

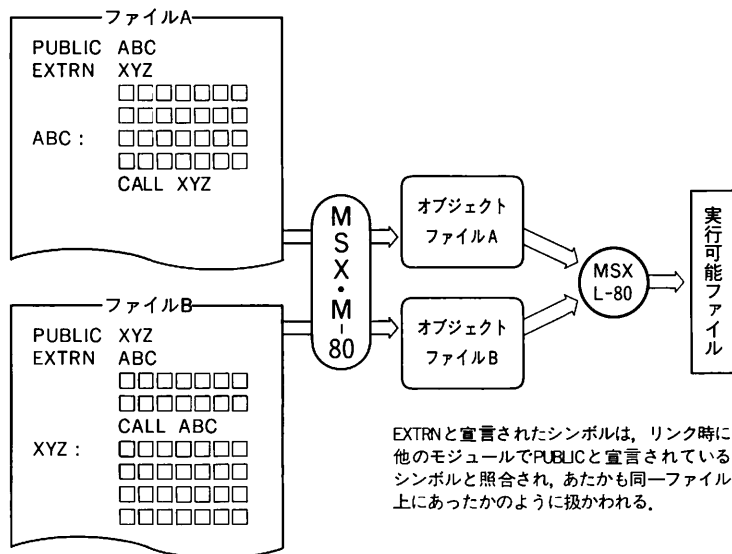



図4.4 PUBLICとEXTRNの考え方

このように，リンク・ローダは外部シンボルの連結を行うことで，各モジュールの関連付けを行います．実は，アセンブラによって完全に処理されるのは内部シンボルだけで，外部シンボルはシンボル名のままでリンク・ローダに渡されるようになっているのです．ここで注意してほしいのは，M-80が出力するリロケータブル・ファイル中の外部シンボルは，プログラムで使っていたシンボルの先頭の6文字だけであるということです．結局，外部シンボルとして有効なシンボル名は先頭から6文字までということになります．実際にプログラムを組む場合にはこの点を考慮しなければなりません．

また，これらの外部シンボルは，擬似命令のPUBLICやEXTRNで宣言する代わりにシンボルやラベルに直接“::”(PUBLIC宣言の代わり)や，“##”(EXTRN宣言の代わり)の記号を付けることによって宣言することもできます(図4.5)．

ファイルA
ABC::
CALL XYZ##
ファイルB
CALL ABC##
XYZ::
まったく同様に
開発を行える
ファイルA'
PUBLIC ABC
EXTRN XYZ
ABC:
CALL XYZ
ファイルB'
PUBLIC XYZ
EXTRN ABC
CALL ABC
XYZ:

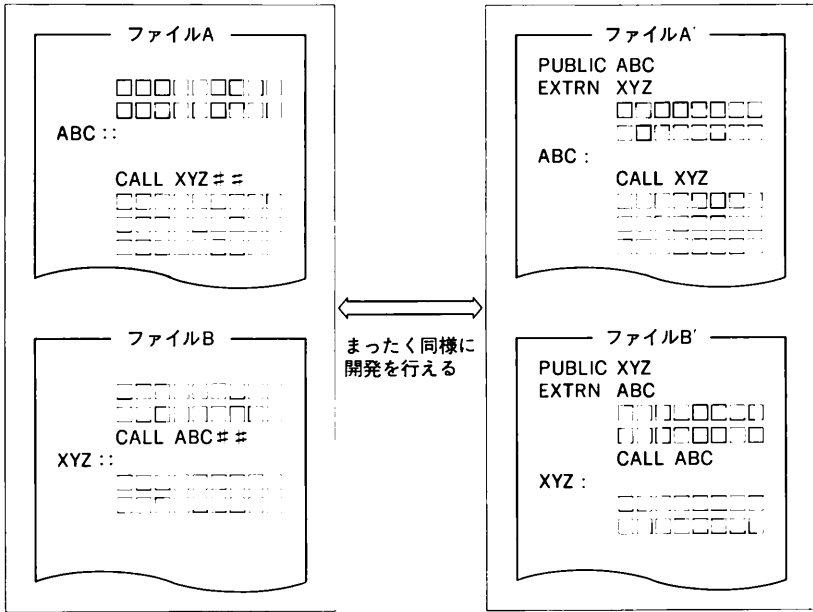
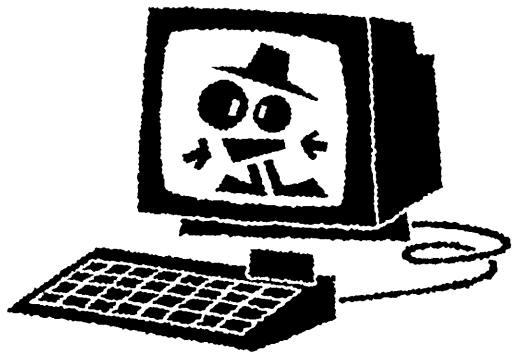



図 4.5　“::”と“##”を用いる

4 3 コード領域とデータ領域

1章で作ったサンプル・プログラムでは，ソース・プログラムが1つだけでしたから，疑似命令の“ASEG”と“ORG”を用いてアセンブルして生成するコードを入れる場所（アドレス）を決めていました．実はそれぞれの命令の意味は次のようなものだったのです．

ASEG ……… これ以降のプログラムは絶対モードであるという宣言

ORG 100H …… プログラムが置かれるアドレスを100H番地にする

ここで，この絶対モードとは，プログラムの配置を特定のアドレスに固定するというモードのことです．この結果プログラムは100H番地から配置されるわけです．

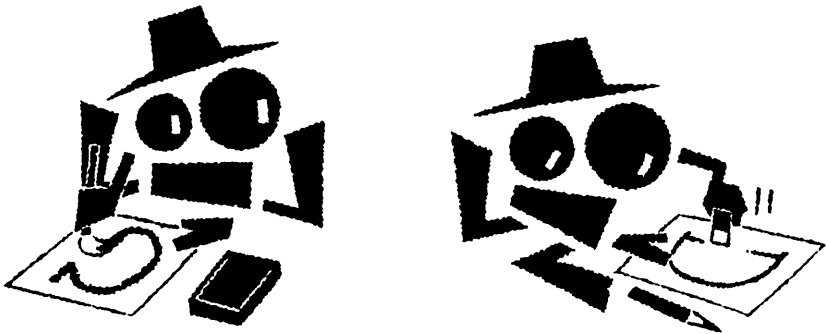
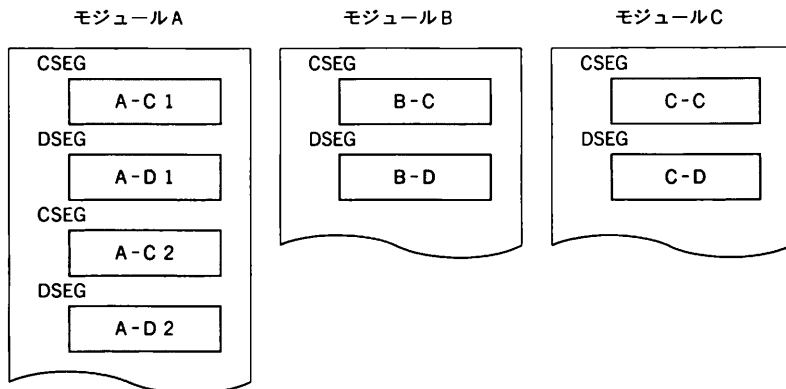
これに対して，モジュール別のプログラムを作る場合には，アセンブル時にその格納されるアドレスを決めることはできません．逆に，どんなアドレスにでも置けるので，モジュールに分けられるということもできます．この場合には“ASEG”を用いることはできません．その代わりにM-80の疑似命令“CSEG”と“DSEG”を用いることになります．これらの宣言をソース・プログラムの先頭（ニーモニックより前）にしておくことで，モジュール内のコードは**リロケータブル（再配置可能の意味）**となります．このときには，“ORG”は使う意味がないということはおわかりでしょう（リロケータブルなモジュールなのですから）．なおCSEGとDSEGの使い分けですが，次のようになっています．

● CSEG (Code−relative)

これ以降はコード領域であり，この領域のコードはプログラムの実行時には書き換えられないという意味です．これ以降の部分は，たとえばROM上に置くことができます．主にプログラム領域や中身の変わらない定数領域に使います．

● DSEG (Data−relative)

これ以降はデータ領域であり，この領域のコードは書き換えられる可能性があるということを指しています．この部分は，RAM上に置かなくてはなりません．主にプログラムのデータ領域に使います．

この使い分けが効果を発揮するのは，プログラム本体をROMに焼き付けたいといったような場合です．つまり，これらのモード指定の使い分けは，プログラムのROM化のためということができます．ただ，MSX-DOSの外部コマンドを作る場合には，プログラム全体がRAM上にロードされるため必要ありません．CSEGとDSEGの使い分けは面倒くさいと思うのならば，プログラム全体をCSEGで書くようにしてもかまいません．しかし，プログラムの意味をはっきりさせるためにも，CSEGとDSEGを正しく使い分ける方がよいでしょう．

プログラムがCSEGとDSEGで書き分けてあっても，リンク・ロード時にアドレスの指定をしない場合には，モジュール名を指定した順に読み込みます．ただし，各モジュール単位では，DSEGの部分とCSEGの部分がそれぞれまとめられて，DSEGの部分，CSEGの部分の順にコードがロードされます（図4.6）．

上に示す内容の各モジュールで下に示すコマンドを実行する

A>L80␣A, B, C, A/N/E

すると，下図のようにリンク・ロードされる

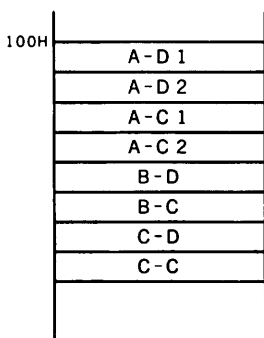



図4.6 /N/Eオプション以外付けない場合のリンク・ロード

一方，リンク・ロード時に次のようなオプションを付けると，コード部とデータ部のアドレス関係をソース・プログラムとは異なったものにすることもできます．

／P：〈番地〉　コード部をロードする先頭番地を指定する（16 進数）
／D：〈番地〉　データ部をロードする先頭番地を指定する（16 進数）

A＞L80␣／P:300, ／D:2000, A, B, C, A／N／E ⏎

図4.6と同じ各モジュールについて，上に示すコマンドを実行すると，下図のようにリンク・ロードされる

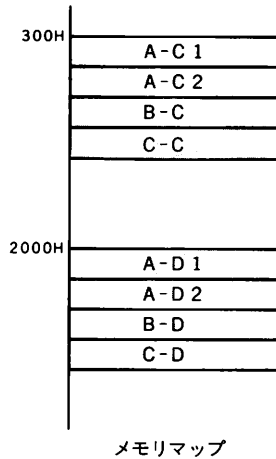



図 4.7　／P,／D オプションを付けてリンク・ロード

ソース・プログラムが１つだけの場合，プログラムはソースの先頭から実行するようになっていました．しかし，モジュールに分けたプログラムでは，どのモジュールからでも実行できるわけではありません．また，１つのモジュールのなかでもコード部とデータ部のアドレスを変えることができるのですから，どのモジュールのどのアドレスから実行させたいのかをアセンブル／リンク・ロード時に設定しておかなくてはなりません．この実行アド

レスを設定する方法は，最初に実行したいモジュールのソース・プログラムの，最後のEND文(アセンブルの終了を示す疑似命令)を以下のように書いておきます.

```
END    <式>
```

そうするとL-80は，生成するマシン語プログラムの先頭アドレスの100H番地から102H番地に，<式>で示されるアドレスにジャンプする命令を書き込みます．このようにしてプログラムの実行開始アドレスを設定します（図4.8).

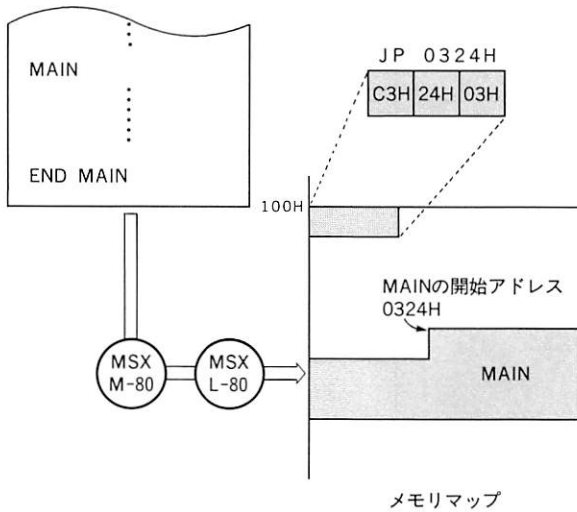



図4.8 プログラムの実行開始アドレスの設定

4 4 時計プログラムを作る

それでは，モジュール化の手法を使ってプログラムを作ってみることにしましょう．前にもいいましたが，モジュール化の真価が問われるのは，大規模なプログラムにおいてです．しかし，ここでいきなりそんなに大きなプログラムを作ることはできませんから，時刻を表示するという簡単なプログラムをサンプルとして取り上げて，プログラムをモジュールに分けて作成することにしました．

プログラムの方針

ひとくちに時刻を表示するプログラムを作るといってもいろいろ決めることがあります．まず問題となるのが，時刻を表示する方法，つまり，アナログにするのかデジタルにするのかということです．しかし，アナログの時計では針の表示をどのように行うかが問題になります．できるだけ簡単なプログラムを作るという理由から，デジタル表示をすることにします．しかし，普通のデジタル表示では当り前すぎるので16進数のデジタル時計を作ることにします．これならば，前に作った16進数の値を表示するサブルーチンを使うことができます．

次に問題となるのが，時刻を表示してからどうするかということです．つまり，表示したらすぐにプログラムを終了してMSX-DOSに戻るようにするか，それともずっと時刻を表示し続けるようにするかということです．ここでは，プログラムが実行されてから何かキーが押されるまでは，時刻を表示し続けるという仕様にします．プログラムの流れを示すと図4.9のようになります．

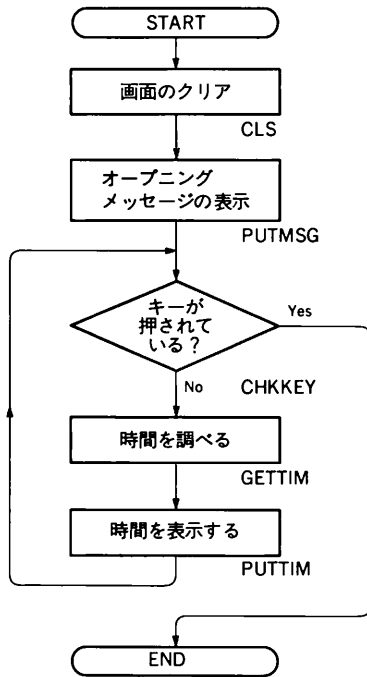



図 4.9　時計プログラムのフロー

これ以外にも具体的な画面構成についても考えなくてはなりません．画面をどのようにするかという点にもプログラマのセンスが出てきますし，実際の使い勝手もずいぶん変わってくるものです．ここで考えることは，「いかに時刻を見やすくするか？」ということと，「いかにプログラムを簡単にできるようにするか？」ということです．また，いろいろな画面モードでもちゃんと表示できるような配慮も必要となります．

いろいろな条件をいいましたが，自分のためのプログラムでは，これらの条件にこだわる必要はありません．オリジナリティあふれる画面を作ってみるのもよいでしょう．ただし，本書ではあとでちょっと手を加える関係上図 4.10 に示すような画面構成に決定します．

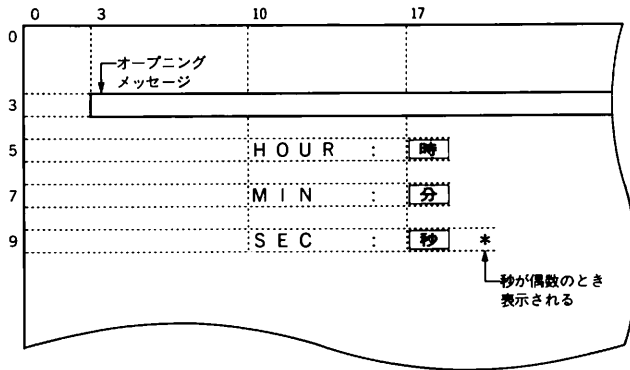



図 4.10 時計プログラムの画面構成

モジュール構成の決定

プログラムをどのように分けるかということ，逆にいうとモジュールごとにどのような機能を持たせるかということも問題になります．極端な話それぞれのサブルーチンごとにモジュールに分けるという方法もあります．そのようにしてもよいのですが，プログラムのファイル管理が大変になってしまうので，ここではプログラムを，①時計機能の中枢部と②入出力の下請け部分の2つのモジュールに分けることにします．

①モジュール1 “CLMAIN.MAC”

MAIN --- メインルーチン
GETTIM --- 時刻読み込みルーチン
PUTTIM --- 時刻表示ルーチン

②モジュール2 “CLSUB.MAC”

PUTMSG --- 文字列表示ルーチン
CLS --- 画面クリアルーチン
CHKKEY --- キー入力検出ルーチン
PUTNUM --- 数字表示ルーチン

ここで，それぞれのモジュール／ルーチン間の関係は図 4.11 のようになっています．

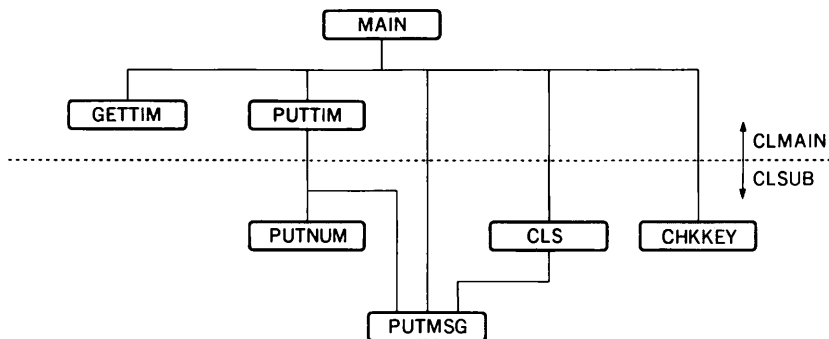



図 4.11 各ルーチン間の関係

必要となるシステムコール

このプログラムのなかでも，いくつかのMSX-DOSのシステムコールを使うことになります．各モジュールのなかのサブルーチンの機能とシステムコールの一覧表を見比べるとだいたいの見当がつくと思います．それぞれの機能と使い方を表 4.1 に示します．

No.	機　能	入　力	出　力
02H	コンソールへの1文字出力	Eレジスタに文字コード	な　し
09H	文字列出力	DEレジスタに文字列の先頭アドレス	な　し
2CH	時刻の獲得	な　し	Hレジスタに時 Lレジスタに分 Dレジスタに秒 Eレジスタに1/100秒

表 4.1 時計プログラムに必要なシステムコールの機能と使い方

いずれのファンクションも，Cレジスタにファンクション番号を入れ，レジスタやメモリを設定し 0005H 番地を呼び出せばよいという点では共通しています．

エスケープ・シーケンス

CRT 画面の制御は，画面を操作するプログラムには欠かすことができないものといえるでしょう．単に文字を端から表示するだけでは，魅力的な画面を構成することは難しいものです．実際，私たちが作ろうとしている時計のプログラムのなかでも，画面の消去(クリア)，カーソルの移動，カーソルのON／OFF などを使います．BASIC ではそれらの機能を実行するためにCLS 文や，LOCATE 文などを使って行うようになっています．MSX-DOSでは，これらの機能を実行する専用のシステムコールは用意されていないのですが，これらの機能が使えないはずがありません．

○カーソル移動	
<ESC>A	カーソルを上に移動
<ESC>B	カーソルを下に移動
<ESC>C	カーソルを右に移動
<ESC>D	カーソルを左に移動
<ESC>H	カーソルをホームポジションに移動
<ESC>Y<Y座標＋20H><X座標＋20H>	カーソルを(X,Y)の位置に移動
○編集，削除	
<ESC>j	画面をクリア
<ESC>E	画面をクリア
<ESC>K	行の終わりまで削除
<ESC>J	画面の終わりまで削除
<ESC>L	1行挿入
<ESC>M	1行削除
○その他	
<ESC>x4	カーソルの形を'■'にする
<ESC>x5	カーソルを消す
<ESC>y4	カーソルの形を'_'にする
<ESC>y5	カーソルを表示する

表 4.2　エスケープ・シーケンス表

MSX-DOSでは、画面の制御を実現するために、特殊な文字列**エスケープ・シーケンス**(ESCape sequence)を用います.このエスケープ・シーケンスはエスケープ文字(ASCIIコードの1BH)とそれに続く文字列によって表されます.この特殊な画面制御用の文字列をコンソール出力する(システムコールを用いる)ことで、画面の制御を行うのです.MSXで使うことができるエスケープ・シーケンスを**表4.2**にあげておきます.

スタックの設定

プログラムのなかでのデータの一時退避，サブルーチン呼び出しの際の戻りアドレスの退避などに**スタック領域**は欠かすことができないものです.2章で作成したサンプル・プログラム“KEYCODE.MAC”では，このスタック領域の設定についてとくに何も考えていませんでした.問題が起こることはまずないのですが，プログラムのなかでスタック領域を設定しておく方がより安全なプログラムとなります.

このスタック領域をどこにするかという問題は，実はそんなに簡単なことではありません.たとえば,スタック領域とシステムのワークエリアが重なってしまうと，スタックを操作することでワークエリアに保存してある内容が変化してしまいますし，その逆も起こりかねません.また，スタック領域をROM上に設定するとデータを書き込むことができなくなります.結局，TPAの後ろからスタック領域を取り，プログラムと重ならないようにするというのが，MSX-DOS上のプログラムでのスタック設定の定石です.

そこで，SP(スタック・ポインタ)の値をどのようにしてTPAの最後に設定するかという方法が問題になりますが，システムコール(0005H番地コール)の飛び先がTPAの直後のアドレスであるという性質を利用して次のようにします.

```
LD      SP,(0006H)
```

プログラムの先頭でこのように宣言しておくことで，**図4.12**のようにスタック領域を確保したことになります.

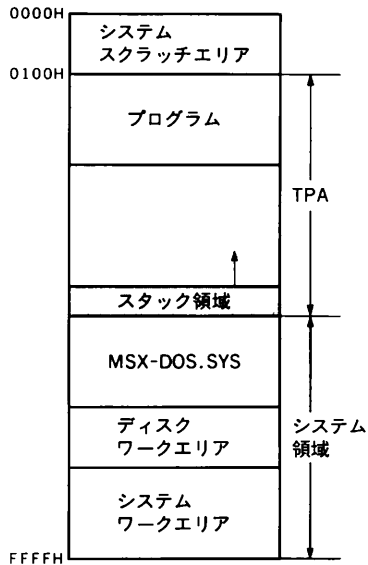



図 4.12 スタック領域の確保

プログラムのなかでSPの値を変えてしまった場合，そのプログラムを終了するときにZ80のRET命令を使って抜け出してはいけないということはおわかりでしょう．

前にあげた方法でスタック領域の確保を行っているプログラムを終了するときには，

```
JP      0000H
```

というようにして0番地に飛ぶか，

```
LD      C,0
CALL    0005H
```

のようにしてシステムリセット（システムコール0H）を呼ぶようにします．

実際のプログラム

これらのことをまとめたプログラムが，次に示すリスト 4.1 “CLMAIN.MAC” とリスト 4.2 “CLSUB.MAC” の 2 つのモジュールです．

```
;***** MSX CLOCK PROGRAM ***** ……………時計プログラムのメイン
;[ MAIN MODULE ]                          モジュール

        EXTRN   CHKKEY ┐
        EXTRN   CLS    │ 他のファイルで定義
        EXTRN   PUTMSG │ されているシンボル
        EXTRN   PUTNUM ┘ を使うという宣言

        .Z80

BOOT    EQU     0000H …………ブート時のジャンプ先
SYSTEM  EQU     0005H …………システムコールの飛び先
ESC     EQU     1BH ……………エスケープ文字の文字コード
EOS     EQU     '$' ……………文字列の終わりを示す(システム
                                コール09Hを利用)

;--- Set Data Macro ---
SETMAC  MACRO   DATPTR,DATREG ……………レジスタの値をメモリに保存する
        LD      A,DATREG                 マクロ．保存するアドレスと保存
        LD      (DATPTR),A               するレジスタ名がパラメータ
        ENDM

;--- Put Data Macro ---
PUTMAC  MACRO   MSGPTR,DATPTR …………データを表示するマクロ．表示する
        LD      DE,MSGPTR                メッセージと保存されているメモリ
                                         の番地をパラメータとする
        CALL    PUTMSG ……………メッセージの表示
        LD      A,(DATPTR) ………数値を取り出す
        CALL    PUTNUM ……………数値の表示
        ENDM

        CSEG ………………………………相対アドレス指定のコード領域

;--- Main Routine --- ……………時計プログラムのメイン・ルーチン
MAIN:
        LD      SP,(SYSTEM+1) ……スタックポインタの設定
        CALL    CLS …………………画面の消去(外部で定義されている)
        LD      DE,MSGOPN
        CALL    PUTMSG ……………オープニング・メッセージの表示
MLOOP:
        CALL    CHKKEY …………キーが押されているかを
        OR      A                調べる(外部で定義)
```

```
        JP      NZ,BOOT ··········キーが押されていればプログラムを終了
        CALL    GETTIM ············そうでなければ現在の時間を調べる
        CALL    PUTTIM ············現在の時間を表示する
        JR      MLOOP ··············何かキーが押されるまで繰り返し
MSGOPN:                                 ······カーソルの移動の文字列
        DB      ESC,'Y',3+20H,3+20H········（エスケープ・シーケンス）
        DB      'MSX CLOCK IS RUNNING NOW!'··············タイトル
        DB      ESC,'x5' ······カーソルを消す（エスケープ・シーケンス）
        DB      EOS ··············文字列の終わり

;--- Get Time & Set Work --- ··············時刻を調べてワークエリア
GETTIM:                                    に保存するルーチン
        LD      C,2CH
        CALL    SYSTEM ············現在時刻を知るシステムコール
        SETMAC  SEC,D ··············秒を保存
        SETMAC  MIN,L ··············分を保存
        SETMAC  HOUR,H ············時を保存
        RET

;--- Print Time --- ··············ワークエリアを参照し時刻を
PUTTIM:                          表示するルーチン
        PUTMAC  MSGHOU,HOUR ············時間の表示
        PUTMAC  MSGMIN,MIN ··············分の表示
        PUTMAC  MSGSEC,SEC ··············秒の表示
        LD      A,(SEC) ··············秒を取り出す
        LD      DE,MSGODD ··········秒が奇数ならMSGODD
        AND     1
        JR      NZ,PUTT1
        LD      DE,MSGEVN ··········秒が偶数ならMSGEVN
PUTT1:
        CALL    PUTMSG ················DEレジスタの示す文字列を表示
        RET

MSGHOU: DB      ESC,'Y',5+20H,10+20H,'HOUR : ',EOS···ガイドラインのメッセージ
MSGMIN: DB      ESC,'Y',7+20H,10+20H,'MIN  : ',EOS （カーソル移動のエスケー
MSGSEC: DB      ESC,'Y',9+20H,10+20H,'SEC  : ',EOS  プ・シーケンスを含む）

MSGODD: DB      '   ',EOS ｝
MSGEVN: DB      '  *',EOS ｝奇数秒／偶数秒のときのメッセージ

        DSEG························データ領域

;--- Work Area --- ··············GETIMで取り込まれた時刻が保存される領域

SEC:    DS      1 ··············秒
MIN:    DS      1 ··············分
HOUR:   DS      1 ··············時

        END     MAIN ··············100H～102H番地に設定されるジャンプ先の登録
```

リスト 4.1 CLMAIN.MAC

```
;***** MSX CLOCK PROGRAM *****  ……時計プログラムのサブ・
;[ SUB MODULE ]                   モジュール(16進表示版)

        PUBLIC  CLS     }
        PUBLIC  PUTMSG  } 他のファイルからも参照される
        PUBLIC  CHKKEY  } 外部シンボルの宣言
        PUBLIC  PUTNUM  }

        .Z80

SYSTEM  EQU     0005H  ……システムコールの飛び先
ESC     EQU     1BH    ……エスケープ文字の文字コード
EOS     EQU     '$'    ……文字列の終わりを示す(シス
                           テムコール09Hを利用)

SYSCALL MACRO   FUNCNO ……システムコール・マクロ
        LD      C,FUNCNO
        CALL    SYSTEM
        ENDM

        CSEG  ……相対アドレス指定(リロケータブルな
                  モジュールにするため)
;--- Clear Screen ---  ……画面消去ルーチン
CLS:
        LD      DE,CLSMSG
        CALL    PUTMSG  ……文字列表示ルーチンの呼び出し
        RET
CLSMSG: DB      ESC,'j',EOS  ……画面消去の文字列
                               (エスケープ・シーケンス)
;--- Print Message (DE = pointer) ---  ……DEレジスタで示される文字
PUTMSG:                                   列を表示する
        SYSCALL 09H  ……システムコール(文字列出力)
        RET

;--- Check Key is Pushed ? ---  ……キーが押されたかどうかを調べる
;[OUT]  A = 0 (Not Pushed) / 0FFH (Pushed)
CHKKEY:
        SYSCALL 0BH  ……システムコール(コンソールの状態を調べる)
        RET

;--- Print Number of Acc (Hexadecimal) ---  ……Aレジスタの値を16進数表示
PUTNUM:                                         するルーチン(もうおなじみの
        PUSH    AF                              はず)
        REPT    4
        RRCA
        ENDM
        CALL    PUTN1
        POP     AF
PUTN1:
        AND     0FH
```

```
        ADD     A,'0'
        CP      '9'+1
        JR      C,PUTN2
        ADD     A,'A'-10-'0'
PUTN2:
        LD      E,A
        SYSCALL 02H
        RET

        END ……………サブ・モジュールの終わりでは
                   アドレスの指定はしない
```

リスト 4.2 CLSUB.MAC

この2つのモジュールのアセンブル，リンク・ロード，実行までの一連のようすを，図 4.13 の実行例で示しましょう．

```
A>M80 =B:CLMAIN.MAC ⏎……………CLMAIN.MACをアセンブルして
                              CLMAIN.RELを作成
No Fatal error(s)

A>M80 =B:CLSUB.MAC ⏎……………CLSUB.MACをアセンブルして
                             CLSUB.RELを作成
No Fatal error(s)

A>L80 B:CLMAIN,B:CLSUB,B:CLK/N/E ⏎……………CLMAIN.RELとCLSUB.REL
                                           をリンクしてCLK.COMを作成
MSX.L-80  1.00  01-Apr-85  (c) 1981,1985 Microsoft

Data    0103    01E6    <  227>

42550 Bytes Free
[0106   01E6        1]

A>B:CLK ⏎……………完成したCLK.COMを実行

                  ⇩ 画面がクリアされる

   MSX CLOCK IS RUNNING NOW!

         HOUR : 10

         MIN  : 2A

         SEC  : 3B█……………偶数秒のときには"*"が表示される
```

図 4.13 時計プログラムの開発と実行

プログラムの改良──10進表示にする──

とりあえず，16進数の時計は完成しました．このままでも別にかまわないのですが，たいていの人は，16進数では違和感を感じるのではないでしょうか．この時計プログラムを10進数の表示に直したものも作っておきましょう．時刻の表示を10進数に直すためだけにプログラム全体を書き直すという必要はありません．“CLSUB.MAC”のなかのPUTNUMルーチンを書き直せばよいのです．このルーチンを書き直した新しいモジュール“CLSUB2.MAC”をリスト4.3に示します．

```
;***** MSX CLOCK PROGRAM *****
;[ SUB MODULE 2 ]

        PUBLIC  CLS
        PUBLIC  PUTMSG
        PUBLIC  CHKKEY
        PUBLIC  PUTNUM

        .Z80

SYSTEM  EQU     0005H
ESC     EQU     1BH
EOS     EQU     '$'

SYSCALL MACRO   FUNCNO
        LD      C,FUNCNO
        CALL    SYSTEM
        ENDM

        CSEG

;--- Clear Screen ---
CLS:
        LD      DE,CLSMSG
        CALL    PUTMSG
        RET
CLSMSG: DB      ESC,'j',EOS
```

時計プログラムのサブ・モジュール(10進表示版)
10進表示版と異なるのは，数字表示ルーチン“PUTNUM”だけなのでそれ以外の部分については16進表示版のリストを参照してください

```
;--- Print Message (DE = pointer) ---
PUTMSG:
        SYSCALL Ø9H
        RET

;--- Check Key is Pushed ? ---
;[OUT]  A = Ø (Not Pushed) / ØFFH (Pushed)
CHKKEY:
        SYSCALL ØBH
        RET

;--- Print Number of Acc (Decimal) --- ……………Aレジスタの内容(0～255)を
PUTNUM:                                          10進数で表示するルーチン
        LD      D,1ØØ
        CALL    PUTSUB…………100の位の表示
        LD      D,1Ø
        CALL    PUTSUB…………10の位の表示
        ADD     A,'Ø'
        LD      E,A
        SYSCALL Ø2H ……………… 1の位の表示
        RET
PUTSUB:………………Aの値からDの値を引くことができるだけ引き，引いた回数(割り算の商)を
        LD      E,Ø ……E＝引いた回数(最初は0)    Dの桁の値として表示する
PUTS1:                                             また，余りの値はAレジスタ
        CP      D ………………AとDを比較            で返される
        JR      C,PUTN ……Aの方が小さければPUTN
        INC     E ………………引いた回数を1増やす
        SUB     D ………………AからDを引く
        JR      PUTS1………繰り返し
PUTN:
        PUSH    AF ……………Aの値(引き算された余り)は保存
        LD      A,'Ø'
        ADD     A,E
        LD      E,A
        SYSCALL Ø2H …………引いた回数を数字キャラクタに直して表示
        POP     AF ……………余りを取り出す
        RET

        END
```

リスト4.3 CLSUB2.MAC

このモジュールをアセンブルして得られる"CLSUB2.REL"と"CLMAIN.MAC"をアセンブルした"CLMAIN.REL"(これはすでに作られているはず)をリンク・ロードすることで,10進数の時計プログラムができます.その作成と実行のようすは図4.14のようになります.

```
A>M8Ø =B:CLSUB2 ↵ ·········CLMAIN.RELはすでに作ってある
                            ので,ここではCLSUB2.MACをア
No Fatal error(s)           センブルするだけでよい
                                               CLMAIN.RELと
A>L8Ø B:CLMAIN,B:CLSUB2,B:CLK2/N/E ↵ ··········CLSUB2.RELを
                                               リンクしてCLK2.
                                               COMを作成
MSX.L-8Ø  1.ØØ  Ø1-Apr-85  (c) 1981,1985 Microsoft

Data    Ø1Ø3    Ø1F4    <  241>

42536 Bytes Free
[Ø1Ø6   Ø1F4        1]

A>B:CLK2 ↵ ·············CLK2.COMを実行

                           ⇩ 画面がクリアされる
   MSX CLOCK IS RUNNING NOW!

          HOUR : Ø16
          MIN  : Ø23    10進数表示になり,
                        わかりやすくなった
          SEC  : ØØ4  *
```

図4.14 10進数表示にする

この時計プログラムの場合,プログラムをモジュール分割してあったため,プログラムの一部を修正(16進数表示を10進数表示に変更)してもプログラム全体をアセンブルし直す必要がなかったわけです.

5章

システムコールの活用

―ファイルの入出力を行う―

これまでの章では，アセンブリ言語の基礎，M-80，L-80の機能と使用法についてひととおり説明してきました．本章では，これらの知識を活かして実践的なプログラミング・テクニック，とくにDOS上でのプログラミングには欠かせないファイルの入出力について考えていきたいと思います．

ファイルを扱うための基本的な機能は，他の入出力をするときと同様，MSX-DOSのシステムコールに用意されています．これらのシステムコールを組み合わせて使えばプログラムのなかで自由にファイルを扱うことができるのですが，そのためにはいくつか知らなくてはならない知識があります．ここではまずファイル操作のために必要な知識を説明します．次に，さまざまなファイル操作を行うためには，どのようにシステムコールを組み合わせて用いればよいのかということについて，簡単なプログラムを紹介しながら見ていくことにします．本章の最後では，まとめとしてファイルの内容を暗号化するプログラムを作ることにしましょう．

5 1 ファイルの操作

ファイルとファイル名

まず，ファイルに対する操作について説明する前に，プログラムのなかでファイルを操作するという立場から，ファイルそのものについて見直してみることとしましょう．

ディスクなど外部の記憶装置に蓄えられているデータを入出力するためには，さまざまな指定をしなくてはなりません（たとえば，何ドライブの第何トラックの第何セクタなど）．また，一度に扱えるデータの大きさは自由に決められるものではなく，ディスク装置によってセクタ・サイズという固有の値に限られています．ディスクのデータを取り扱いやすくするために，「意味のあるデータ」ごとにひとまとまりとして扱えるようにしたもの，これが**ファイル** (file) です．ファイルのなかにはデータやプログラムを自由に保存できます．

このファイルは，MSX-DOSによって管理されています．つまり，アプリケーション・プログラムがファイルを扱うようにMSX-DOSに命令（システムコール）すると，MSX-DOSがディスク装置の細かい制御をし，読み書きを行うのです．

このようなMSX-DOSの下働きのおかげで，私たちは読み書きしたいデータがディスクのどの場所にあるかということをいっさい知る必要はありません．私たちはただ「このファイルを読め！」などと命令してやればよいのです．この命令の際には，目的のファイルを他のファイルと区別しなければなりません．このためファイルに名前を付け（ファイル名），その名前でファイルを指定するという方法をとります．システムコールの際にファイル名をどのように指定するかについては，のちほど詳しく見ていくことにします．

ファイルの管理

では，この指定されたファイルに対してどのような操作が考えられるでしょうか？　もちろんファイルはデータを格納するものですから，中身のデータの読み書きは最も重要かつ基本的な操作です．また，普通1枚のディスクのなかにはいくつものファイルがはいっているので，それらを管理するための操作も必要になります．それらの操作としては次のようなものが考えられます．

- ファイル名の変更
- ファイルの削除
- ファイルの検索

ただし，ここでファイルの検索というのは，指定したファイルが存在するかどうかを調べるという意味のものです．

これらの操作は，いずれもMSX-DOSの内部コマンドREN，DEL，DIRなどで行うことができます．もちろん，アプリケーション・プログラムからでも，これらのファイル管理の操作ができるようにシステムコールが用意されています．

ファイルの読み書き手順

システムコールを利用してファイルを読み書きする場合の手順を考える前に，BASICでのファイルのアクセスの方法を思い出しておくことにしましょう．

BASICでは，ファイルの内容を読み書きする前にOPEN文を使ってファイルを使用することをコンピュータに宣言しておかなくてはなりませんでした．たとえば，“SAMPLE.DAT”というファイルを先頭から順に（シーケンシャルに）書き込むようにオープンする場合は，次のように宣言します．

```
OPEN "SAMPLE.DAT" FOR OUTPUT AS #1
```

この宣言以後は，ファイル番号（この場合は#1）を用いて書き込みを行うことができます．たとえば“Text Output!”という文字列を出力する場合は，次のようにしてやればよいのです．

```
PRINT #1, "Text Output!"
```

そしてファイルに対しての書き込みを終えた場合には，普通は次のようにファイルをクローズしてファイルへのアクセスを終了します．

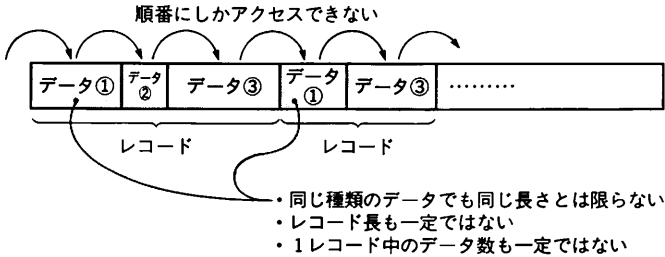
```
CLOSE #1
```

このようなファイルに対する操作は，システムコールを用いて行う場合でも基本的な手順は同じです．これらの手続きを要約すると次のようになります．

①ファイルの使用開始手続き
②入出力操作
③ファイルの使用終了手続き

ファイルの読み書きの方法

BASICでも，ファイルの読み書きには大きく分けて2つの方法がありました．**シーケンシャル・アクセス**と**ランダム・アクセス**です．シーケンシャル・アクセスは，ファイルの先頭から順にデータを読む（書く）というものです．これに対して，ランダム・アクセスは，ファイルをレコードという単位ごとに任意の順で読む（書く）というものです．ここでレコードというのは，1回の読み書きで取り扱うデータの集まりのことです（図5.1）．

・シーケンシャルアクセス

・ランダムアクセス

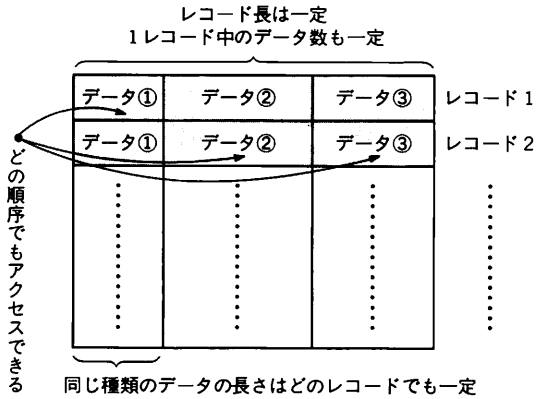



図 5.1　シーケンシャル・アクセスとランダム・アクセス

MSX-DOSのシステムコールによるファイル読み書きについても，このシーケンシャル・アクセスとランダム・アクセスの両方ができるようになっています．ただ，このシーケンシャルとランダムというファイルのアクセス法の区別の前に，大きく分けて2系統のシステムコールが存在します．

MSX-DOSというMSX用のOSは，8ビットCPU用の主流のOSであったCP／M-80上のソフトウェアがそのまま動くように作られています．このため，ファイル・アクセスのためのシステムコールもCP／Mと互換性のあるもの，いわゆる**CP／M方式**が用意されています．いっぽう，MSX用に独自に機能を拡張した**ランダム・ブロック・アクセス**と呼ばれるものも用意されています．

● CP／M 方式

CP／M ではファイル・アクセス法として，シーケンシャル・アクセスとランダム・アクセスの 2 種類に分かれています．ここで 1 つのレコードの大きさ（レコード・サイズ）はディスク装置の性質を反映して，128 バイトに固定されています．それ以外のレコードを扱いたいときには，アプリケーション・プログラムの内部でそのためのルーチンを用意しなくてはなりません．

● ランダム・ブロック・アクセス

この MSX-DOS 独自のシステムコールは大変に有能です．その特徴の 1 つは，レコードサイズをプログラムで任意の値（1～65535 バイト）に取ることができることです．したがって，ファイルのなかの 1 バイトを 1 レコードにもできますし，128 バイトを 1 レコードとして扱うこと（CP／M 方式）もできます．また，ファイルがそれほど大きなものでなければ，ファイル全体を 1 レコードとして扱うことさえできるのです．もう 1 つの特徴は，1 度のアクセスで複数のレコードを読み書きできることです．

このアクセス法はその名前が示すとおりランダム・アクセスです．ただ，アクセスするたびにアクセスするレコード番号が示される領域（ランダム・レコード）を次のレコード番号に更新するようになっているので，最初にランダム・レコードを設定すれば，あとはシーケンシャルにファイルをアクセスすることも可能になっているのです．

このように MSX-DOS では，2 系統のファイル・アクセス法が用意されています．しかし，その機能を比べてみればわかりますが，MSX 独自のランダム・ブロック・アクセスは，CP／M 方式のアクセスの機能を含んでいます．CP／M 方式は，CP／M で作られた多くのソフトウェア資産を MSX で継承できるようにするため，あるいは CP／M でも MSX-DOS でも共通に動くプログラムを作れるようにするために用意されたシステムコール群でなのです．私たちは，MSX-DOS で動くプログラムを作ろうとしているのですから，このランダム・ブロック・アクセスのシステムコールを取り扱っていくことにします．

5 2 ディスク・アクセスのための設定

FCB（ファイル・コントロール・ブロック）

システムコールでファイルをアクセスするときには、どのファイルに対するアクセスであるのかを指定しなければなりません．そのため，アクセスするファイルのファイル名を指定してやる必要があります．MSX-DOS のシステムコールでは，BASIC のように文字列でファイルを指定するのではなく，FCB（File Control Block）と呼ばれる 37 バイトの領域で指定します．この領域にはファイルを扱うために必要となる情報が格納されます．この情報というのは，アプリケーション・プログラムと DOS のシステムとの間でやり取りするためのものだけではなく，DOS のシステム内部でそのファイルを扱うために必要な情報も含まれています．また，この FCB は同時にアクセスするファイルの数だけ必要で，この領域をアプリケーション・プログラムが用意する必要があります（図 5.2）．

では，これらの各フィールドの内容を見ていきましょう．フィールド名の次の括弧に囲まれた数値は，FCB の先頭からのバイト数を示しています．また，私たちが用いるランダム・ブロック・アクセスを行う際に，パラメータとしてアプリケーション・プログラムで設定する必要があるフィールドには“＊”を付けて他のフィールドと区別しておきます．

・**ドライブ番号（0）　＊**

ファイルの存在するドライブ名を示す

（0：デフォルトドライブ，1：A ドライブ，2：B ドライブ，…，8：H ドライブ）

FCB
先頭からのバイト数
↓

バイト	内容
0	ドライブ番号
1 ⋮ 11	ファイル名 ファイル名前部……………………8バイト 拡張子………………………………3バイト
12 13	カレント・ブロック ファイルの先頭から現在のブロックまでのブロック数
14 15	レコード・サイズ 1〜65535
16 ⋮ 19	ファイル・サイズ 1〜4294967296
20 21	日付 ディレクトリと同形式
22 23	時刻 ディレクトリと同形式
24	デバイスID
25	ディレクトリ・ロケーション
26 27	ファイルの開始クラスタ番号
28 29	最後にアクセスしたクラスタ番号
30 31	ファイルの開始クラスタからの相対位置 最後にアクセスしたクラスタのファイルの先頭からのクラスタ数
32	カレント・レコード
33 ⋮ 36	ランダム・レコード ファイルの先頭から何番目のレコードであるか 通常は最後にランダムアクセスしたレコードが格納されている

図 5.2 FCB の構造

- **ファイル名前部（1〜8）　＊**

  最大 8 文字まで．8 文字に満たない部分は，スペース (20H) で埋めておく

- **ファイルの拡張子（9〜11）　＊**

  最大 3 文字まで．3 文字に満たない部分は，スペースで埋めておく

- **カレント・ブロック（12〜13）**

  シーケンシャル・アクセスのときに，参照中のブロック番号を示す

・**レコードサイズ（14～15）**　＊

ランダム・ブロック・アクセスを行うときの1つのレコードの大きさを示す（バイト単位）

・**ファイルサイズ（16～19）**

ファイルの大きさがバイト単位で設定される

・**日付（20～21）**

最後にファイルに書き込みを行った日付が設定される（図 5.3）

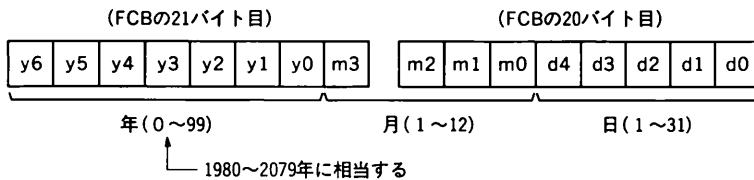



図 5.3　日付の書式

・**時刻（22～23）**

最後にファイルに書き込みを行った時刻が設定される（図 5.4）

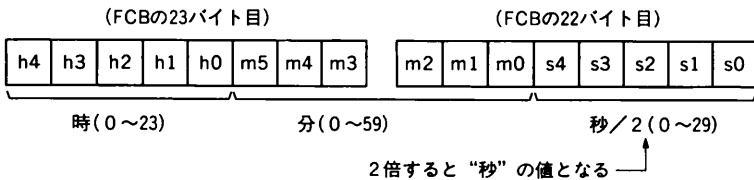



図 5.4　時刻の書式

・**デバイス ID（24）**

周辺装置／ディスクファイルの区別が設定される

・**ディレクトリロケーション（25）**

何番目のディレクトリ・エントリにあるファイルかが設定される

・**先頭クラスタ（26～27）**

ファイルの先頭のクラスタ番号が設定される

- **最終アクセス・クラスタ (28～29)**
  最後にアクセスされたクラスタ番号が設定される
- **最終アクセス・クラスタの先頭クラスタからの相対位置 (30～31)**
  最後にアクセスされたクラスタのファイル内での相対番号が設定される
- **カレント・レコード (32)**
  シーケンシャル・アクセスのとき，参照中のレコード番号を示す
- **ランダム・レコード (33～36)　＊**
  ランダム・アクセス (CP／M 互換) およびランダム・ブロック・アクセスの際，アクセスしたいレコードの番号を示す

このFCBを用いてファイルを指定してアクセスしますが，前にも説明したように，ファイルにアクセスするためにはまずファイルをオープンしなくてはなりません．ファイルをオープンするとFCBの内容が変化します．このようすを図5.5に示します．

このように，ドライブ名とファイル名だけしか設定されていないFCBから，ディスクのディレクトリ領域に記された情報を用いて完全なFCBに直すということがシステムコールの「ファイルのオープン」なのです．

いっぽう，ファイルに書き込みを行うと，それに応じてFCBの各フィールドの値も変化します．この更新されたFCBの情報をディスクのディレクトリ領域に書き戻しておかないと，次からのファイル・アクセスのときに，ディレクトリの情報と，実際のファイルの内容がくい違ってしまいます．この更新されたFCBの情報をディレクトリ領域に書き戻すという手続きがシステムコールの「ファイルのクローズ」だったのです．このためにファイルを読んだだけのときには，ファイルをクローズする必要がないのです．

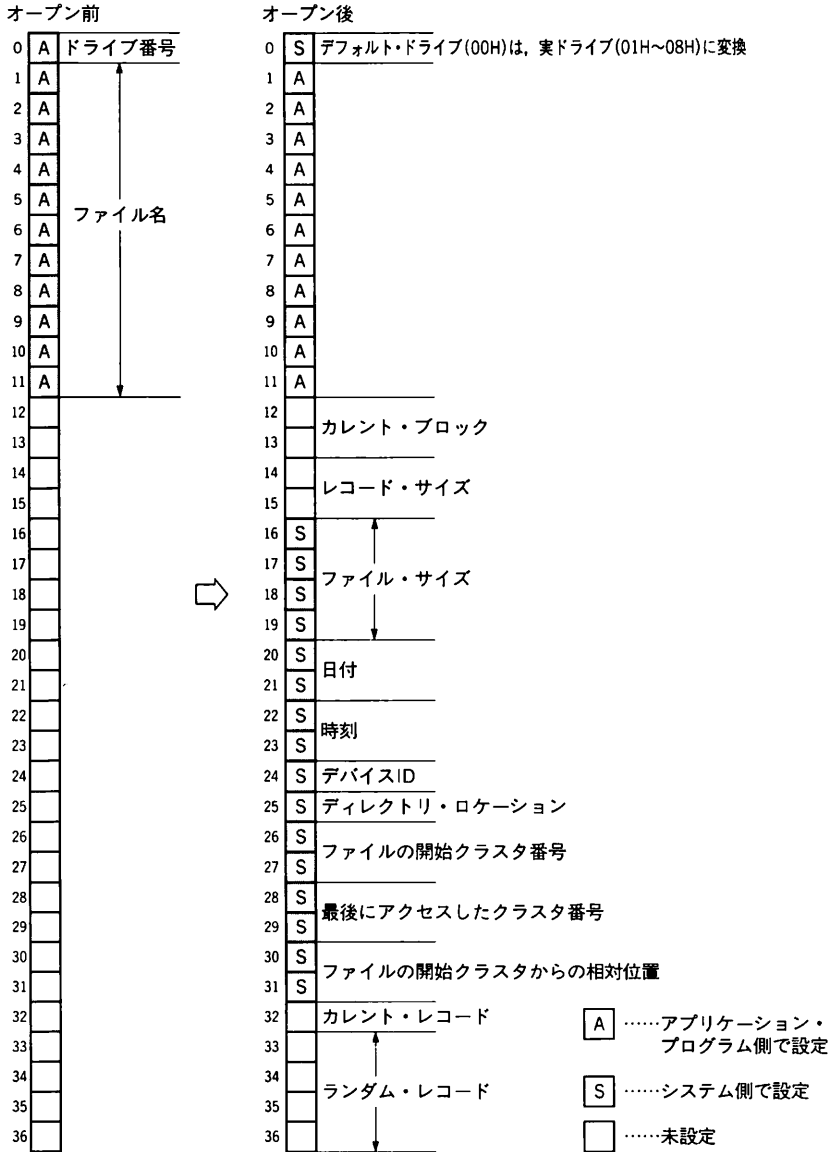
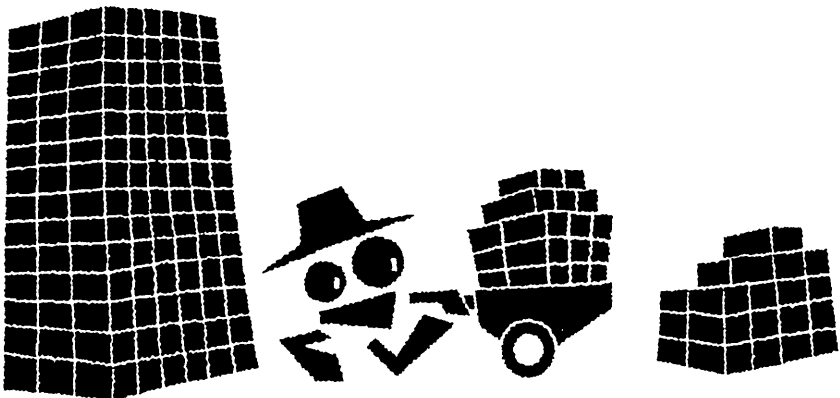



図 5.5　オープン前後の FCB の内容

DTA（ディスク転送アドレス）

ディスクの入出力では，他の周辺機器の入出力と違って，一度に複数バイトのデータをやり取りします．そのため，ディスクの入出力は「バッファ」(データをやり取りするためのメモリ領域)を介して行われます．つまり，ディスクからデータを読むときにはこのバッファにデータがはいり，ディスクにデータを書くときにはこのバッファの内容が書かれるのです．このバッファの大きさは，CP／M 互換のファイル・アクセスでは 128 バイト固定ですが，ランダム・ブロック・アクセスでは，〈レコードサイズ〉×〈一度にアクセスするレコード数〉となります．いずれにしろ，この大きさのメモリ領域をアプリケーション・プログラムが用意する必要があります．

MSX-DOS では，このバッファとして使用されるメモリ領域の先頭アドレスを DTA (Disk Transfer Address) と呼びます (CP／M にならって「DMA アドレス」と呼ぶこともあります)．アプリケーション・プログラムを立ち上げた時点，つまり外部コマンドが読み込まれて実行されたときには DTA は 0080H です．この DTA は，システムコール 1AH「DTA の設定」で任意の値に変更することができるようになっています．

5 3 ファイル操作とシステムコール

ファイルの操作は，いくつかの種類に分けられます．ここではそれぞれのファイル操作のときのシステムコールを使う手順を示し，簡単なサンプル・プログラムを見てもらうことにします．

ファイル名の変更

システムコール17Hを使うことで，ファイルの名前を変更することができます（図5.6）．

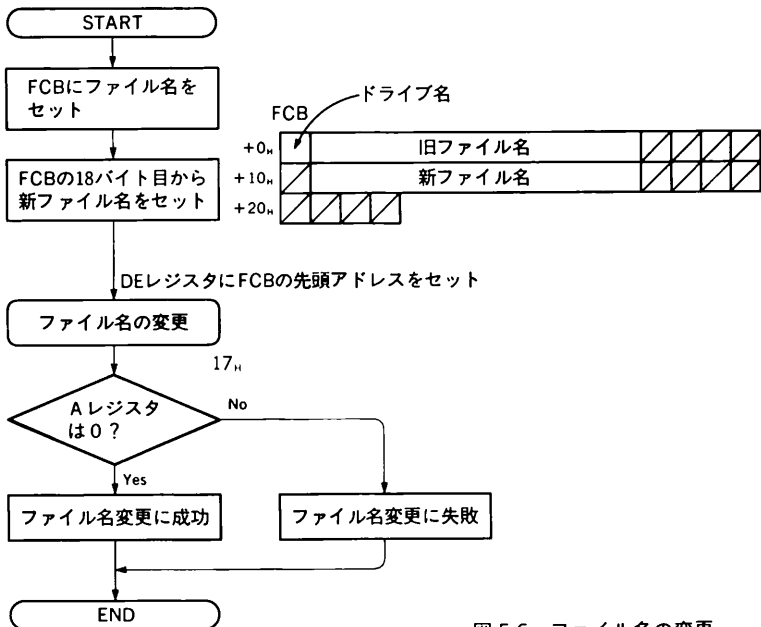



図5.6　ファイル名の変更

ファイル名の変更では，DOS の REN コマンドと同様にワイルドカード文字“？”を用いることができます．ただ“＊”は使うことができませんから，図 5.7 に示すようにプログラムのなかで適当な数の“？”に展開しなくてはなりません．

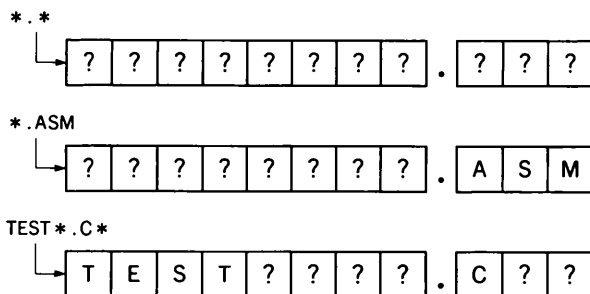



図 5.7　ワイルドカード文字の展開

サンプル・プログラムとして，“.TXT”という拡張子の付いたファイルを“.DOC”という拡張子に付け変えるプログラムをリスト 5.1 に示します．

```
; RENAME FILE SAMPLE

        .Z80

        ASEG
        ORG     100H

        LD      DE,FCB
        LD      C,17H
        CALL    0005H·············システムコール(ファイル名変更)
        JP      0000H
FCB:····································変更対象ファイルのFCB
        DB      0··············ドライブ番号(0：カレント・ドライブ，1：Aドライブ，…)
        DB      "????????TXT"·············変更前のファイル名
        DS      5,0                      (FCB+1～FCB+11)
        DB      "????????DOC"·············変更前のファイル名
        DS      9,0                      (FCB+17～FCB+27)

        END
```

リスト 5.1　ファイル名の変更サンプル・プログラム

ファイルの削除

FCBに削除するファイルのドライブ名とファイル名を設定して，システムコール13Hを呼ぶことで，指定したファイルが削除されます(図5.8)．このときファイル名にワイルドカード文字として“？”を用いることができ，まとめてファイルを削除するということ(ある意味では大変危険なことですが)もできるようになっています．ただし，ファイルの削除でもファイル名の変更と同様に“*”を用いることはできませんから，複数の“？”に展開する必要があります．

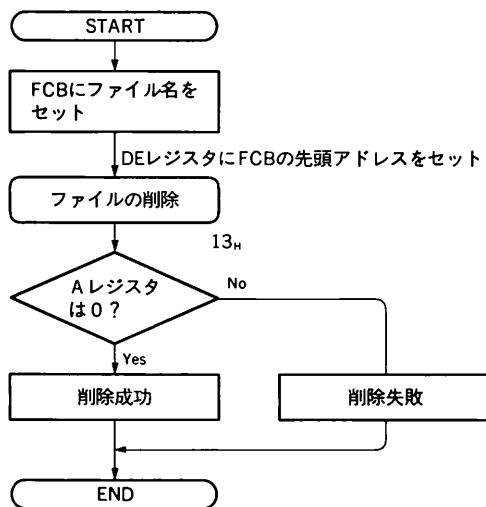



図5.8　ファイルの削除

例として“SAMPLE.DOC”というファイルを削除するプログラムをリスト5.2に示しますが，このプログラムの内容はファイルを削除するシステムコール（13H）を呼び出しているだけです．

このプログラムでは，あらかじめ決まったファイルを削除していますが，あとでコマンドラインの内容を読む方法を説明するときに指定したファイル

を削除できるように拡張します.

```
; DELETE FILE SAMPLE

        .Z80

        ASEG
        ORG     100H

        LD      DE,FCB
        LD      C,13H
        CALL    0005H……………システムコール(ファイル削除)
        JP      0000H
FCB:……………削除するファイルのFCB
        DB      0 ……………ドライブ番号
        DB      "SAMPLE  DOC"……………ファイル名(FCB+1~FCB+11)
        DS      25,0

        END
```

リスト 5.2 ファイル削除サンプル・プログラム

ファイルの検索

ファイルを検索し該当するファイルが存在するかどうかは、そのファイルをオープンする(システムコール 0FH)ことでも調べられます. というのは, ファイルがオープンできないときにはそのファイルがディスクに存在しないことを示しているからです.

しかしファイルのオープンでは, ワイルドカード文字“?”を利用して, まとめてファイルを指定することはできません. たとえば“????????.COM”と指定してして, 拡張子が“.COM”であるファイルを探すというような場合です. このようなファイルの検索は, システムコールの 11H と 12H を組み合わせることで実現されます. つまり, システムコールの 11H で最初のファイルを検索しておき, さらにシステムコール 12H を用いて次のファイルを探すという手順を取るのです (図 5.9).

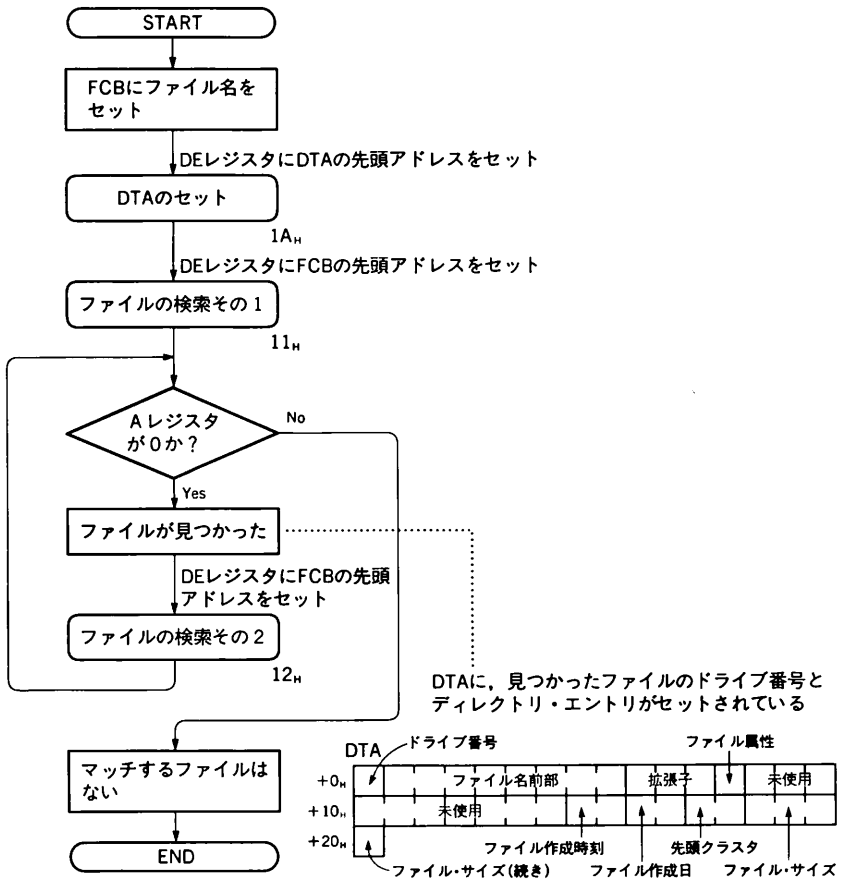



図5.9　ファイルの検索

このシステムコールの組み合せを使って，デフォルト・ドライブの“.COM”という拡張子を持つファイルのファイル名を画面に表示するプログラムをリスト5.3に示します．“.COM”という拡張子を持つファイルを探すのですから，結果としてデフォルト・ドライブの中にある外部コマンドの一覧が表示されることになります．

```
; SEARCH FILE SAMPLE

        .Z80

SYSCALL MACRO   FUNCNO……………システムコール・マクロの定義
        LD      C,FUNCNO
        CALL    0005H
        ENDM

SCALLA  MACRO   FUNCNO,ARG……………システムコール・マクロ(DEレジスタ
        LD      DE,ARG                   に値を与える)の定義
        SYSCALL FUNCNO
        ENDM

PUTCHR  MACRO   CHR……………1文字出力マクロの定義
        LD      E,CHR
        SYSCALL 02H
        ENDM

        ASEG
        ORG     100H

        SCALLA  1AH,DTA……………DTAの設定
        SCALLA  11H,FCB……………最初のファイルを検索
LOOP:
        OR      A
        JR      NZ,EXIT……………該当するファイルがなければEXIT
        CALL    P_NAME……………見つかったファイル名を表示
        SCALLA  12H,FCB……………次のファイルを検索
        JR      LOOP
EXIT:
        JP      0000H……………ブートしてDOSに戻る

P_NAME:……………DTAに設定されたファイル名を表示するルーチン
        LD      HL,DTA+1………HL＝ファイル名の先頭
        LD      B,11……………B＝表示する文字数
P_N1:
        PUSH    BC    ┐
        PUSH    HL    │
        PUTCHR  (HL)  │
        POP     HL    ├ アドレスHLからB文字を表示
        POP     BC    │
        INC     HL    │
        DJNZ    P_N1  ┘
        PUTCHR  0DH   ┐
        PUTCHR  0AH   ┘ 改行する
        RET
```

```
FCB:……………検索対象ファイルのFCB
        DB      Ø……………ドライブ番号
        DB      "????????COM"……………ファイル名
        DS      25,Ø

DTA:    DS      33 ……………システムコール11Hまたは12Hに
                          よって，見つかったファイルのド
        END               ライブ番号とディレクトリ・エン
                          トリ情報が入れられる
```

リスト 5.3 ファイルの検索サンプル・プログラム

ファイルを読む

ひとくちにファイルを読むといっても，シーケンシャルな読み込みもあれば，ランダムな読み込みもあります．ここでは扱いが簡単なシーケンシャルな読み込みを取り上げます．ただし，MSX のランダム・ブロック・アクセスを使うときには，シーケンシャル・アクセスでもランダム・アクセスでも本質的な違いはなく，ランダムな読み込みの場合には読み込む前にランダム・レコードを設定する必要があるだけです．

ファイルを読む場合は，これまでのファイルを管理する操作のようにシステムコール１つで実行できるわけではありません．ファイルをオープンするという操作もはいってくるからです．具体的には，まずファイルのオープン（システムコール 0FH）を行ってから，読み込みアドレスの設定（システムコール 1AH）や，FCB に読み込むためのレコード・サイズや，ランダム・レコードを設定するという作業が余分に必要となります．

また，ファイルを読んだあとも，ファイルが正しく読み込めたかどうかを調べて，その結果に応じた処理をしなくてはなりません．たとえば，ファイルの終わりに達したために，ファイルを読もうとしたサイズ分読み込めない場合にどうするかが問題となるのです．

これらのことを考慮すると，シーケンシャルにファイルを読む手順は図 5.10 のようになります．

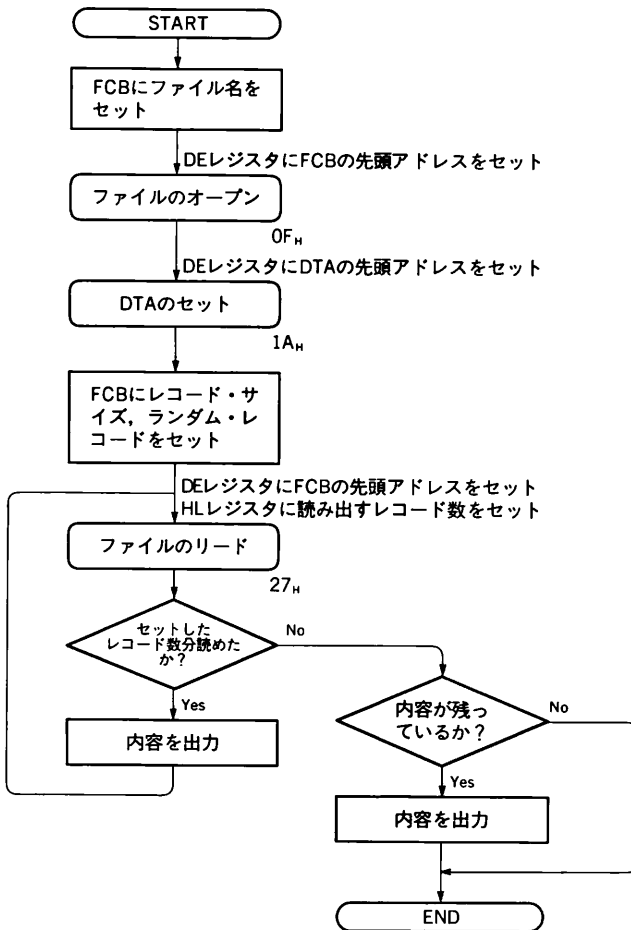



図 5.10　ファイルを読む

例として“SAMPLE.DOC”というファイルを読み，その内容を画面に表示するプログラムをリスト 5.4 に示します．

```
; READ FILE SAMPLE

        .Z80

BUFSIZ  EQU     512 ……………ファイルを読むときのバッファの大きさ

SYSCALL MACRO   FUNCNO………システムコール・マクロの定義
        LD      C,FUNCNO
        CALL    0005H
        ENDM

SCALLA  MACRO   FUNCNO,ARG ……………システムコール・マクロ(DEレジ
        LD      DE,ARG                  スタに値を渡す)の定義
        SYSCALL FUNCNO
        ENDM

        ASEG
        ORG     100H

        SCALLA  0FH,FCB ……………ファイルのオープン
        OR      A
        JR      NZ,NOFILE ………ファイルがなければNOFILEへ
        SCALLA  1AH,DTA ……………あれば，DTAの設定
        LD      HL,1
        LD      (FCB+14),HL ……レコード・サイズの設定(ここでは1バイト)
        LD      HL,0
        LD      (FCB+33),HL
        LD      (FCB+35),HL………読む先頭のレコード番号(ここでは0)
LOOP:
        LD      HL,BUFSIZ ………読み込むレコード数
        SCALLA  27H,FCB ……………ランダム・ブロック読み込み
        OR      A
        JR      NZ,NOTFULL ….読み込みが不完全
        CALL    PUTDTA …………読み込んだ内容を表示
        JR      LOOP ……………ファイルが終わるまで繰り返し
NOTFULL:……………読み込みが不完全な場合(指定したレコード数読み込むことができなかった)
        LD      A,H
        OR      L
        CALL    NZ,PUTDTA …………… 1レコード以上読み込んだの
        JR      EXIT                    ならば，内容を表示
NOFILE:……………ファイルが存在しない場合
        SCALLA  09H,M_NOFL ……………文字列M_NOFLを表示
EXIT:
        JP      0000H ……………ブートしてDOSに戻る
M_NOFL: DB      "No File !$"

PUTDTA:……………DTAの内容を表示するルーチン(HLバイト)
        LD      DE,DTA
```

```
PUTD:
        PUSH    HL     ┐
        PUSH    DE     │
        LD      A,(DE) │
        LD      E,A    │
        SYSCALL 02H    │
        POP     DE     │ DEレジスタの示すアドレス
        POP     HL     │ からHL文字分表示する
        INC     DE     │
        DEC     HL     │
        LD      A,L    │
        OR      H      │
        RET     Z      │
        JR      PUTD   ┘

FCB:……………読み込むファイルのFCB
        DB      0 ……………ドライブ番号
        DB      "SAMPLE  DOC"……………ファイル名(FCB+1～FCB+11)
        DS      25,0

DTA:    DS      BUFSIZ ……………読み込むファイルのバッファ領域

        END
```

リスト 5.4 ファイル読み込みサンプル・プログラム

ファイルに書く

ファイルにデータを書くといってもいろいろな場合が考えられますが，あるデータのはいったファイルを作成するという場合は，まず何もデータのはいっていないファイルを作成し，そのファイルにデータを書くという手順を取らなくてはなりません．ファイルを作成するシステムコール 16H では，すでに存在するファイルと同じ名前のファイルを作ることができませんから，同じ名前のファイルはあらかじめ消しておくことが要求されます．結局ファイルを作成するためには図 5.11 のような手順が必要となります．

例として“DONMENU.DOC”というファイルを作り，そのファイルに適当な文字列を書き込むプログラムをリスト 5.5 に示します．

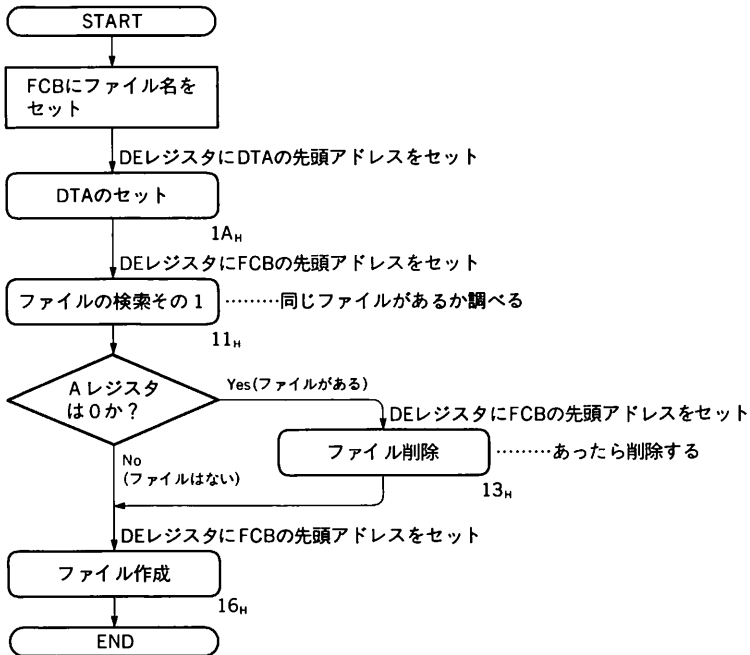



図5.11　ファイルを作成する

```
; CREATE AND WRITE FILE SAMPLE

        .Z80

CR      EQU     0DH  ┐
LF      EQU     0AH  ├ 定数値の定義
EOF     EQU     1AH  ┘

SCALLA  MACRO   FUNCNO,ARG ……………システムコール・マクロ(DEレジ
        LD      DE,ARG                スタに値を渡す)の定義
        LD      C,FUNCNO
        CALL    0005H
        ENDM

        ASEG
        ORG     100H

        SCALLA  11H,FCB ……………最初のファイル検索
        OR      A                 (このときのDTAは0080H)
```

```
        JR      NZ,NOFILE ··········ファイルがなければNOFILEへ
        SCALLA  13H,FCB ··············ファイル削除
        OR      A
        JR      NZ,CANTMK···········削除できなければ, CANTMKへ
NOFILE:
        SCALLA  16H,FCB ··············ファイルの生成
        OR      A
        JR      NZ,CANTMK···········失敗すればCANTMKへ
        SCALLA  ØFH,FCB ··············生成したファイルのオープン
        OR      A
        JR      NZ,CANTMK ···········失敗すればCANTMKへ
        SCALLA  1AH,DATPTR ·········DTAアドレスを設定
        LD      HL,1
        LD      (FCB+14),HL········レコード・サイズの設定
        LD      HL,Ø                （ここでは, 1バイト）
        LD      (FCB+33),HL········先頭のレコード番号(ここでは0)
        LD      (FCB+35),HL
        LD      HL,DATEND-DATPTR ········HL＝レコード数
        SCALLA  26H,FCB ··············ランダム・ブロック書き込み
        SCALLA  1ØH,FCB ··············ファイルのクローズ
        JR      EXIT
CANTMK: ··············ファイルが作れない場合
        SCALLA  Ø9H,M_CTMK ··············文字列M_CTMKを表示
EXIT:
        JP      ØØØØH ··············ブートしてDOSに戻る
M_CTMK: DB      "Can't Write!$"

FCB:··············読み込むファイルのFCB
        DB      Ø ··············ドライブ番号
        DB      "DONMENU DOC"··············ファイル名(FCB＋1 ~FCB＋11)
        DS      25,Ø

DATPTR: ··············書き込むデータの先頭
        DB      "Una Don",CR,LF
        DB      "UnaTama Don",CR,LF
        DB      "Ten Don",CR,LF
        DB      "Katsu Don",CR,LF
        DB      "Kaika Don",CR,LF
        DB      "Gyuu Don",CR,LF
        DB      "Tanin Don",CR,LF
        DB      "Oyako Don",CR,LF
        DB      "Chinese Don",CR,LF
        DB      "Curry Don",CR,LF
        DB      "Egg Don",CR,LF
        DB      EOF
DATEND:··············書き込むデータの終わり

        END
```

リスト 5.5 ファイル作成サンプル・プログラム

5-4 コマンド・ラインの内容を知る

私たちの作成するプログラムはMSX-DOSの外部コマンドという形で実行されるのですが，プログラムにファイル名などのパラメータを与え，それに従って動くようにするにはどうしたらよいでしょうか？

まず考えられるのは，プログラムの実行時に必要となるパラメータをキーボードから入力してもらうというものです．これは，文字入力や文字列入力のシステムコールを使うことで実現できます．

もう1つは，プログラムの実行を指定するときに用いられるコマンド・ラインに，そのプログラムに与えるパラメータも書くというものです．たとえば“NANNO.COM”を実行するときに，次のように“GELATO”，“GARLAND”という文字列と数値623を与えます．

```
A> NANNO  GELATO  GARLAND  623
```

プログラム実行時にパラメータを入力する方法だと，そのコマンドの使い方を知らなくても，プロンプトに従って入力すればよいのですからユーザーが戸惑うことは少ないでしょう．いっぽう，コマンド・ラインにパラメータを書く方法では，コマンドの処理内容と必要とするパラメータの種類と順番を正確に把握していないと，正しく入力できません．

しかし，バッチコマンド（“.BAT”の拡張子が付く）を利用して，キーボードを打つ手間を省きたいときには，コマンド・ラインを利用する方法でなければ，パラメータを自動的に指定できません．結局，まとめて行わせる処理はコマンド・ラインから，間違ってはいけない入力は実行時に直接キーボードから，というように使い分けるとよいのです．

ここでは，コマンド・ラインの内容を知るにはどうすればよいかを見ていきます．MSX-DOSではコマンドを与えられると，まずパラメータを決まった

領域に格納します．そして，プログラム（外部コマンド）をディスクから読み込んで，100H にジャンプ，つまりプログラムに制御を移すというような実行の手順を取ります．プログラムのなかからは，決まった領域を読むことでコマンド・ラインの内容を知ることができるのです．

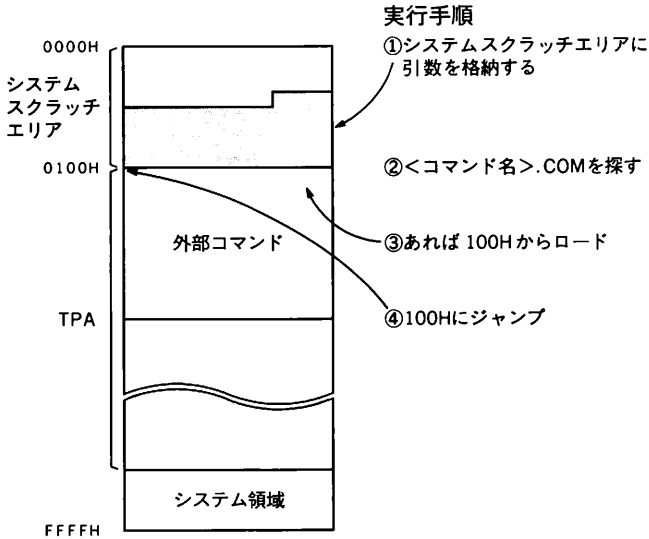



図 5.12 外部コマンド実行までのようす

このコマンド・ラインの内容が格納される領域は，システムスクラッチエリアのなかにあります．システムスクラッチエリアの先頭部には，いくつかのジャンプ命令が並んでいます．0 番地がブート（プログラムを終了してコマンド待ちの状態に戻る）のためのエントリ（入口）ですし，5 番地はシステムコールのためのエントリです．また，RDSLT，WRSLT，ENASLT，CALLF は，インタースロットコールのためのエントリです．このインタースロットコールについては 7 章で解説することにします．INTRPT は割り込みのためのエントリです．0H～5BH まではシステムも使用しているので，不用意にその内容を変更してはいけません（図 5.13）．

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
00H	JP WBOOT					JP BDOS							JP RDSLT			
10H					JP WRSLT								JP CALSLT			
20H					JP ENASLT											
30H	JP CALLF								JP INTRPT							
40H																
50H													←			
60H	デフォルトのFCB　その1												←			
70H	デフォルトのFCB　その2															
80H																
90H																
A0H																
B0H	デフォルトのDTA															
C0H																
D0H																
E0H																
F0H																

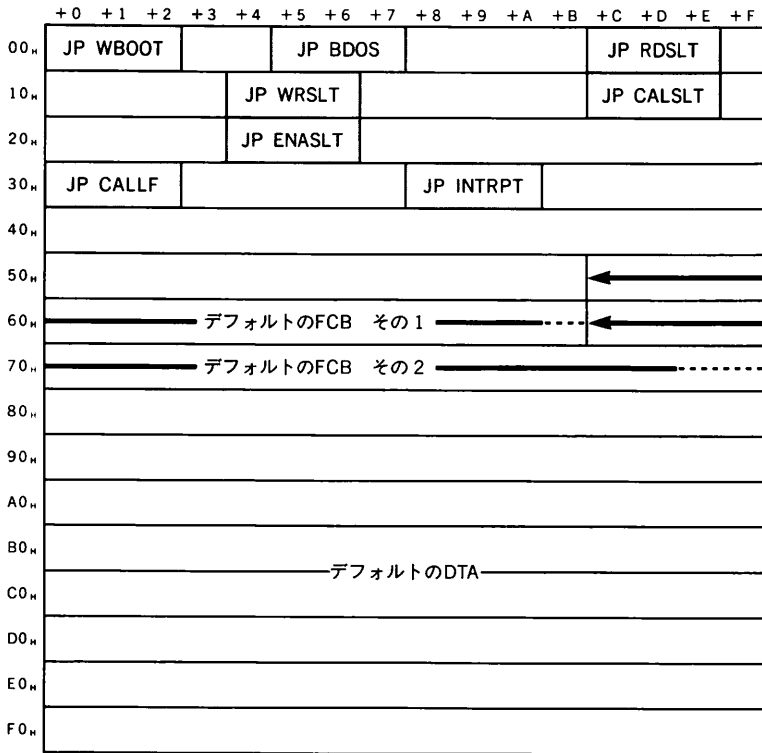



図5.13　システムスクラッチエリア

コマンド・ラインの内容はプログラムが扱いやすいように2つの方法で格納されています．では，この2つの方法についてそれぞれ説明していくことにしましょう．

コマンド・ラインの格納　その1 ── 80H

コマンド・ラインの内容がそのままはいっているのがこの領域です．80Hにはコマンド・ラインに与えられたパラメータの文字数がはいり，81Hからはその内容がはいっています（図5.14）．

A>MYECHO␣MSX-DOS

コマンド・ラインから上のように入力すると，メモリ上には下の図のように格納される

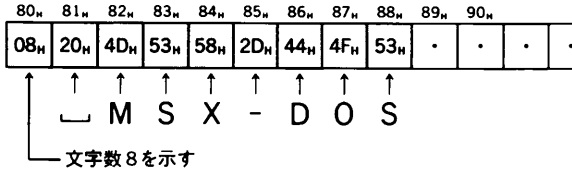



図5.14　コマンド・ラインの格納---その1

コマンド・ラインの内容をそのまま画面に表示するプログラムをリスト5.6に示しますが，その内容は80Hからの領域を表示するというものです．このプログラムと同様な働きをするものとして“MSX-DOS TOOLS”のなかのECHOコマンドがありますから，ここで作るプログラムは“MYECHO”としておきます．

```
;***** MYECHO.MAC *****

        .Z80

DTA     EQU     0080H ···········ディスク転送アドレス(コマンド・
                                 ライン受渡しアドレス)
PUTCHR  MACRO   CHR ·············1文字表示マクロ
        LD      E,CHR
        LD      C,02H
        CALL    0005H
        ENDM

        ASEG
        ORG     100H

        LD      HL,DTA
        LD      A,(HL) ··········A＝コマンド・ラインの文字数
        OR      A
        JR      Z,EXIT ·········もし文字数＝0ならばEXITへ
        INC     HL ··············HL＝コマンド・ラインの内容の先頭
        LD      B,A ·············B＝文字数
```

```
LOOP:
        PUSH    BC
        PUSH    HL
        PUTCHR  (HL)         アドレスHLからの内容をB文字表示
        POP     HL
        POP     BC
        INC     HL
        DJNZ    LOOP
EXIT:
        PUTCHR  ØDH          改行する
        PUTCHR  ØAH
        JP      ØØØØH

        END
```

リスト 5.6　MYECHO.MAC

コマンド・ラインの格納　その 2 ── 5CH と 6CH

80H からの領域は，コマンド・ラインの内容がそのままはいっているため，プログラムしだいで一度にたくさんのパラメータをとることもでき汎用性があります．しかし，一般に DOS 上のプログラムはファイルを扱うものが多く，コマンド・ラインに書かれたファイル名を処理を行う対象ファイルとする場合が多いものです．コマンド・ラインに与えられたパラメータをファイル名であると解釈し，プログラムが作りやすいように加工して格納されるのが，この 5CH と 6CH からの領域です．この領域には最高 2 つのファイル名を格納することができます（図 5.15）．

A>COPY␣A:BLOOM.TXT␣B:BLOOM.BAK

コマンド・ラインから上のように入力すると，メモリ上には下の図のように格納される

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
50_H	·	·	·	·	·	·	·	·	·	·	·	·	01_H	42_H	$4C_H$	$4F_H$
60_H	$4F_H$	$4D_H$	20_H	20_H	20_H	54_H	58_H	54_H					02_H	42_H	$4C_H$	$4F_H$
70_H	$4F_H$	$4D_H$	20_H	20_H	20_H	42_H	41_H	$4B_H$								

図 5.15　コマンド・ラインの格納---その 2

この形式を見ればわかりますが，そのままFCBとして使えるようになっています．したがって，ファイル名を1つしか受け取らないプログラムでは，5CHからをFCBとすることもできます．しかしファイル名を2つ受け取る場合，1つのFCBの大きさは37バイトなのでこの2つの領域（5CHからと6CHから）は重なってしまいます．この場合は，適当な領域にこの内容を転送してから使わなくてはなりません．また，3つ以上のファイル名を取るときには，80Hからの内容をファイル名に直すという作業をアプリケーション・プログラムのなかで行う必要があります．

この領域を読み，指定されたファイルを削除するプログラム（いわゆるDELコマンド）をリスト5.7に示します．

```
;***** MYDEL.MAC *****

        .Z80

BOOT    EQU     0000H……………ブートの飛び先
SYSTEM  EQU     0005H……………システムコール
FCB     EQU     005CH……………デフォルトFCB
DTA     EQU     0080H……………デフォルトDTA

        ASEG
        ORG     100H……………プログラムは100Hから

        LD      A,(DTA)……………A＝コマンドに与えられたパラメータの文字数
        OR      A              :…… パラメータがない(文字数
        JR      Z,NOPARAM……:      ＝0)ならばNOPARAMへ
        LD      DE,FCB……………DE＝デフォルトFCB
        LD      C,13H
        CALL    0005H……………ファイル削除
        JR      EXIT
NOPARAM:……………パラメータがなかった場合
        LD      DE,M_NOPA
        LD      C,09H
        CALL    0005H……………文字列M_NOPAを表示する
EXIT:
        JP      0000H……………プログラムを終了してDOSに戻る
M_NOPA:……………使い方の簡単な説明
        DB      "Usage: mydel <filename>"
        DB      0DH,0AH,'$'

        END
```

リスト5.7 MYDEL.MAC

暗号化プログラム“MYCRYPT”を作る

では，ファイル入出力のまとめとして，ファイルの内容を暗号化するプログラムを作ることにしましょう．暗号化するといっても，もとのファイルに簡単に戻せなくては意味がありませんので，ここでは図 5.16 に示すような手軽な方法を採ることにします．

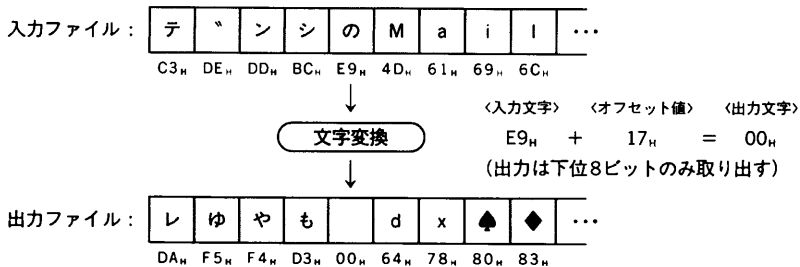



図 5.16　暗号化処理

プログラムの大きな流れは，最初にコマンド・ラインからパラメータとしてオフセット値と入力ファイル名を取り出します．あとはファイルが終わるまでデータを読み，そのデータを 1 文字ずつ変換して，2 つ目のファイル（デフォルト・ドライブの“a.out”）に書くというものです（図 5.17）．

このコマンド“MYCRYPT”の書式は，つぎのように決めます．

MYCRYPT　＜オフセット値＞　＜入力ファイル名＞

ここで＜オフセット値＞は 0～255 の 10 進数で入力してください．

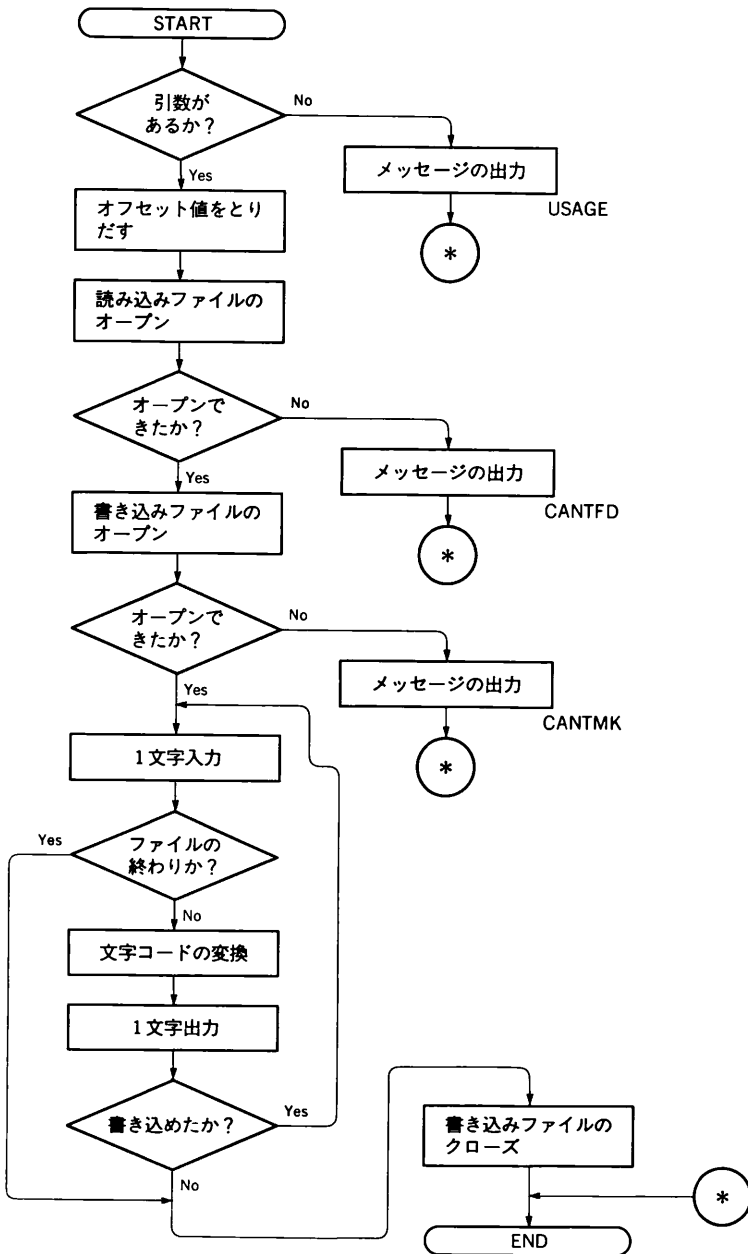



図 5.17 MYCRYPT のフロー

また，1バイトごとに文字を読む（書く）処理についても少し考えておくことがあります．簡単に思い付くのは，レコード・サイズを1バイトとして1レコードずつ読む（書く）という方法です．たしかにこの方法でもちゃんと処理を行ってくれるのでまったく問題はありません．しかし，実際にそのようなルーチンを作ってみると，入出力に非常に時間がかかることがわかります．そこでこのプログラムの1バイト読むというルーチン（FGETC）のなかでは，まずまとまった量のデータをバッファに読み込んでおいて，FGETCルーチンが呼ばれるたびにバッファのなかの1文字ずつを順に取り出すという方法を使います．

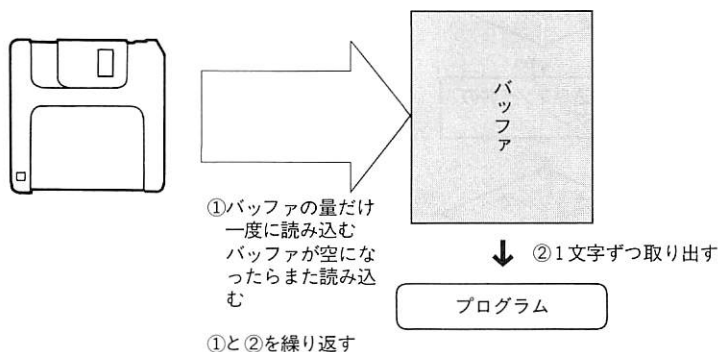



図5.18　バッファを用いた1文字入力

また1文字出力（FPUTC）では，書き込まれたデータを，ひとまずバッファに保存しておいて，書き込むデータが揃うとバッファの内容をディスクに書き込むというようにします．なお，ファイルにデータを書いたあとには，ファイルをクローズしなくてはなりません．このとき，バッファにデータが残っていれば，そのデータをファイルに書き込んでからファイルをクローズします．

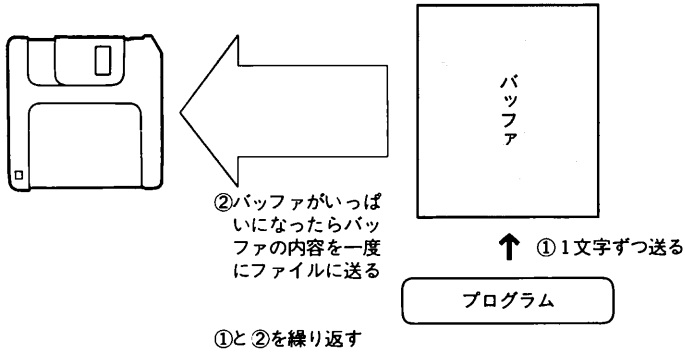



図 5.19　バッファを用いた1文字出力

このようにバッファを用いて入出力を行うと他のルーチンが同じものでも処理速度はずいぶん改善されるのです．これらのことを考慮して実際にプログラムを書くのですが，リストが長くなるため，次に示す2つのモジュールに分けることにしました．

MYCRYPT.MAC　……　メイン・ルーチンおよび暗号化処理
FILE.MAC　……　ファイルの入出力

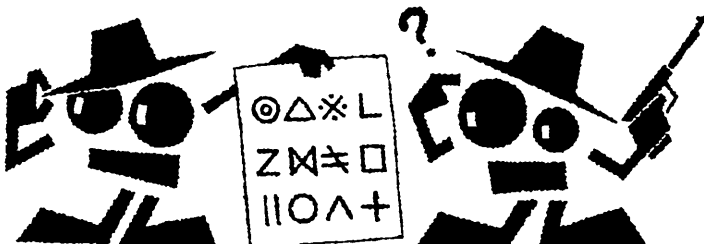
各モジュールのリストをリスト 5.8 とリスト 5.9 に示します．

```
; MY CRYPT : ENCRYPT AND DECRYPT

        EXTRN   FOPENR, FGETC  ┐
        EXTRN   FOPENW, FPUTC  ├ 外部で定義されたルーチン
        EXTRN   FCLOSW         ┘

        .Z80

CR      EQU     0DH
LF      EQU     0AH
EOS     EQU     '$'
SYSTEM  EQU     0005H ……………システムコールの呼び出し先
BOOT    EQU     0000H ……………ブート時の飛び先
FCB2    EQU     006CH ……………デフォルトFCBの2つ目
DTA     EQU     0080H ……………デフォルトDTA
```

```
ERREXT  MACRO   ERRMSG…………………エラー・メッセージを出力した
        LD      DE,ERRMSG          あとでブートするマクロの定義
        LD      C,Ø9H
        CALL    SYSTEM
        JP      BOOT
        ENDM

        CSEG……………………相対アドレス指定

;--- Main Routine ---

MAIN:
        LD      SP,(SYSTEM+1)
        CALL    CHKARG…………………引数のチェック
        OR      A
        JR      NZ,USAGE…………引数が異常なときはUSAGEへ
        CALL    GETOFS…………………オフセット値を取り出す
        LD      DE,FCB2
        CALL    FOPENR…………………読み込みファイルのオープン
        OR      A
        JR      NZ,CANTFD…………オープンできないときはCANTFDへ
        LD      DE,FCBW
        CALL    FOPENW…………………書き込みファイルのオープン
        OR      A
        JR      NZ,CANTMK…………オープンできないときはCANTMKへ
LOOP:
        CALL    FGETC……………ファイルから1文字入力
        JR      C,EXITL…………ファイルの終わりならEXITLへ
        CALL    CNVCHR……………文字コードの変換
        CALL    FPUTC……………ファイルに1文字出力
        JR      C,EXITL…………書き込めなければEXITLへ
        JR      LOOP………………繰り返し

;--- Normal End --- ………………正常な終了
EXITL:
        CALL    FCLOSW……………書き込みファイルのクローズ
        JP      BOOT………………ブートしてDOSに戻る

;--- Abnormal End --- ………………エラーによる異常な終了
USAGE:………………引数がおかしい場合
        ERREXT  M_USAG
M_USAG: DB      "Usage : mycrypt <number> <sour.name>"
        DB      CR,LF,EOS
CANTFD:………………読み込みファイルが見つからない場合
        ERREXT  M_CTFD
M_CTFD: DB      "Can't find sour-file."
        DB      CR,LF,EOS
CANTMK:………………書き込みファイルが作れない場合
```

```
        ERREXT  M_CTMK
M_CTMK: DB      "Cant make dest-file."
        DB      CR,LF,EOS

;--- Write File FCB (FileName = "a.out") --- ……………書き込み
FCBW:   DB      Ø                                           ファイル
        DB      "A       OUT"                               のFCB
        DS      37-12,Ø

;--- Check Command Line Argument --- ……………コマンドの引数の
;[OUT] A = Ø : (normal) / ELSE : (abnormal)  チェック（引数が
CHKARG:                                      あるかないかのチェ
        LD      HL,DTA                       ックをしているだけ）
        LD      A,(HL)
        INC     HL
        OR      A
        LD      A,1
        RET     Z
        XOR     A
        RET

;--- Get Offset Code (Decimal) --- ……………引数からオフセット
GETOFS:                                     値を取り出す
        LD      HL,DTA+1
GETOF1:
        LD      A,(HL)      ┐
        CP      ' '         │
        JR      NZ,GETOF2   ┤ 引数の先頭の空白を読み飛ばす
        INC     HL          │
        JR      GETOF1      ┘
GETOF2:
        CALL    GETNUM ……………引数を数値に変換
        LD      A,D
        LD      (OFFSET),A ………オフセット値に保存
        RET

        DSEG

OFFSET: DS      1 ……………文字コードに加えるオフセット値

        CSEG

;--- Convert Character <Acc> --- ……………文字コードの変換
CNVCHR:                                   （暗号化）
        LD      HL,OFFSET
        ADD     A,(HL)……………A = A + (OFFSET)
        RET
```

```
;--- Get Number <D> from String <HL> --- ··············文字列を10進
GETNUM:                                                 数値に変換
        LD      D,Ø
GETNU1:
        LD      A,(HL)
        INC     HL
        CALL    COD2NUM
        RET     C
        LD      E,A
        LD      A,D  ┐
        ADD     A,A  │
        ADD     A,A  │
        ADD     A,D  ├ D = D * 10 + E
        ADD     A,A  │
        ADD     A,E  │
        LD      D,A  ┘
        JR      GETNU1

;--- Char to Number <Acc> --- ··············文字を10進数値に
;( if Not number_char then CY = 1 )   変換するルーチン
COD2NUM:
        CP      'Ø'
        RET     C··············A<"0"ならCY = 1 で戻る
        CP      '9'+1
        CCF
        RET     C··············A>"9"ならCY = 1 で戻る
        SUB     'Ø'············それ以外のとき
        RET·························A = A - "0", CY = 0 で戻る

        END     MAIN ·········100H番地からのジャンプ命令の
                              生成を指定する(メイン・モジュ
                              ールだから)
```

リスト 5.8 MYCRYPT.MAC

```
; FILE.MAC : File Handling Functions

        PUBLIC  FOPENR, FGETC  ┐
        PUBLIC  FOPENW, FPUTC  ├ 外部から参照できる
        PUBLIC  FCLOSW         ┘ サブルーチン群

        .Z80

SYSTEM  EQU     0005H ………… システムコールの呼び出し先
BUFSIZ  EQU     512 ……………… ファイルバッファの大きさ
                              (読む／書く)

SYSCALL MACRO   FUNCNO ……… システムコール・マクロの定義
        LD      C,FUNCNO
        CALL    SYSTEM
        ENDM

SCALLA  MACRO   FUNCNO,ARG ……………… システムコール・マクロ(DEレジス
        LD      DE,ARG                  タに値を渡すタイプ)の定義
        SYSCALL FUNCNO
        ENDM

        CSEG …………… 相対アドレス指定(リロケータブル
                      なモジュールだから)

;--- File Open for Read --- ……………… ファイルのオープン
;[IN]   DE = FCB                        (FGETCのため)
;[OUT]  A = 0 : (success) / ELSE : (failure)
FOPENR:
        LD      (RD_FCB),DE …… FCBの置かれているアドレスを保存
        CALL    CLRFCB …………… FCBを初期化する
        CALL    OPEN ……………… ファイルのオープン
        LD      HL,0 …………… バッファ上に残っているデー
        LD      (RD_LFT),HL  タ数＝0(まだファイルを読
                             み込んでいないため)
        RET …………… 戻り値はOPENによって決まる

;--- Get Char from File --- ……………… あらかじめFOPENRでオープンした
;[OUT]  CY = 0 (success) ; A = CHAR  ファイルから1文字読み込むルー
;       CY = 1 (EOF)                 チン
FGETC:
        LD      HL,(RD_LFT)
        LD      A,H
        OR      L
        JR      Z,FGETC1 …………… まだ読まれていないデータがバッファ
        DEC     HL                 に残っていなければFGETC1へ
        LD      (RD_LFT),HL ……… 読まれていないデータの数を1減らす
        LD      HL,(RD_PTR) ……… HL＝次に読むデータのポインタ
        JR      FGETC2
```

```
FGETC1:……………バッファにファイルからデータを読み込む
        SCALLA  1AH,RD_BUF …………DTAの設定
        LD      HL,BUFSIZ ……………読み込むレコード数
        SCALLA  27H,(RD_FCB)………ランダム・ブロック読み込み
        LD      A,H
        OR      L
        SCF
        RET     Z ……………ファイルから読み込めたブロック数
                          が0ならばCY=1で戻る
        DEC     HL
        LD      (RD_LFT),HL……バッファ上でまだ読まれていな
                          いデータの数を設定
        LD      HL,RD_BUF ……………次に読むデータのポインタ
                          (バッファの先頭)
FGETC2:
        LD      A,(HL)……………DTAから1文字を取り出す
        INC     HL
        LD      (RD_PTR),HL …次に読むポインタを1つ増やして
        OR      A ……………………CY=0で戻る
        RET
        DSEG

;--- Work Area for Read --- ……………ファイル読み込みのための
                          ルーチンFOPEN, FGETCで
                          用いるワーク領域
RD_FCB: DS      2 ……………FCBのアドレス
RD_LFT: DS      2 ……………バッファ上にあって読まれていないデータの残りバイト数
RD_PTR: DS      2 ……………バッファ上で次にどのデータを読むかを示すポインタ
RD_BUF: DS      BUFSIZ……ファイル読み込み用のバッファ領域

        CSEG

;--- Create File and Open for Write --- ……………ファイルのオープン
;[IN]   DE = FCB                                 (FPUTCのため)
;[OUT]  A = Ø : (success) / ELSE : (failure)
FOPENW:
        LD      (WT_FCB),DE ……FCBの置かれているアドレスを保存
        CALL    CLRFCB ……………FCBの初期化
        SYSCALL ØFH…………………すでに同名のファイルが存在する
        OR      A                 かどうか調べる
        JR      NZ,NOFILE………存在しなければNOFILEへ
        SCALLA  13H,(WT_FCB) …存在すれば,そのファイルを削除
        OR      A
        RET     NZ ……………削除できなければ戻る(A=0)
NOFILE:
        SCALLA  16H,(WT_FCB)……………ファイルを生成
        OR      A
        RET     NZ ……………生成できなければ戻る(A=0)
        LD      DE,(WT_FCB)
        CALL    OPEN……………ファイルのオープン
        LD      HL,BUFSIZ
        LD      (WT_FRE),HL ……………バッファの残り領域はバッファ全体
```

```
        LD      HL,WT_BUF
        LD      (WT_PTR),HL……………次に書き込む位置はバッファの先頭
        RET……………戻り値はOPENによって決まる

;--- Put Char to File --- ……………あらかじめFOPENWしたでオープンした
;[IN]   A = CHAR                      ファイルに1文字書き込むルーチン
;[OUT]  CY = Ø : (success) / 1 : (failure)
FPUTC:
        LD      C,A ……………C＝書き込む文字
        LD      HL,(WT_FRE)
        LD      A,H
        OR      L
        JR      Z,FPUTC1 ……………バッファの残り領域がなければFPUTC1へ
        DEC     HL
        LD      (WT_FRE),HL……………残り領域を1減らす
        LD      HL,(WT_PTR)……………HL＝データを次に書く位置
        JR      FPUTC2
FPUTC1:……………現在のバッファの内容をファイルに書き込む
        PUSH    BC
        SCALLA  1AH,WT_BUF ……………DTAの設定
        LD      HL,BUFSIZ……………書き込みレコード数
        SCALLA  26h,(WT_FCB)………ランダム・ブロック書き込み
        POP     BC
        OR      A
        SCF
        RET     NZ ……………書き込みが失敗なら，CY＝1として戻る
        LD      HL,BUFSIZ-1     ……バッファの残り領域は
        LD      (WT_FRE),HL……     バッファの大きさ－1
        LD      HL,WT_BUF ……………データを次に書くのは
FPUTC2:                           バッファの先頭
        LD      (HL),C ……………バッファにデータを書く
        INC     HL
        LD      (WT_PTR),HL ………次に書く位置を1増やす
        OR      A ……………CY＝0
        RET

;--- File Close for Write --- ……………書き込みファイルのクローズ
FCLOSW:
        SCALLA  1AH,WT_BUF ……………DTAの設定
        LD      HL,(WT_PTR)
        LD      DE,WT_BUF
        OR      A
        SBC     HL,DE ……………HL＝バッファに溜っているデータ数
        JR      Z,FCLOS1………バッファに残っていなければFCLOS1へ
        SCALLA  26h,(WT_FCB)……………ランダム・ブロック書き込み
FCLOS1:
        SCALLA  1ØH,(WT_FCB)……………ファイルのクローズ
        RET

        DSEG
```

```
;--- Work Area for Write --- ……………ファイル書き込みのためのルーチンFOPENW, FPUTC, FCLOSで用いるワーク領域

WT_FCB: DS      2 ……………FCBのアドレス
WT_FRE: DS      2 ……………バッファ上でまだ書かれていない空き領域の大きさ
WT_PTR: DS      2 ……………バッファ上のどのアドレスに次のデータを書くかを示すポインタ
WT_BUF: DS      BUFSIZ ……………ファイル書き込み用のバッファ領域

        CSEG

;--- Clear (FCB+12)~(FCB+36) --- ……………FCBのうちファイル名でない領域を0で消去するルーチン
;[IN]   DE = FCB (Not Modify)
CLRFCB:
        LD      HL,12
        ADD     HL,DE
        LD      B,36-11
CLRF1:
        LD      (HL),Ø
        INC     HL
        DJNZ    CLRF1
        RET

;--- File Open and Set Parameter --- ……………ファイルを開けてFCBを設定
;[IN]   DE = FCB
;[OUT]  A = Ø : (success) / ELSE : (failure)
OPEN:
        PUSH    DE
        SYSCALL ØFH ……………ファイルのオープン(システムコール)
        POP     IX
        OR      A
        RET     NZ ……………失敗ならばAレジスタは0以外で戻る
        XOR     A
        LD      (IX+14),1
        LD      (IX+15),A ……………レコードの大きさ=1バイト
        LD      (IX+33),A
        LD      (IX+34),A
        LD      (IX+35),A
        LD      (IX+36),A ……………先頭レコード番号=0
        RET ……………Aレジスタには0がはいって戻る

        END ……………実行アドレスは書かない(サブモジュールだから)
```

リスト 5.9 FILE.MAC

このプログラムをアセンブルして実行するまでのようすを図5.20に示します．

```
A>M80 =B:MYCRYPT ⏎

No Fatal error(s)

A>M80 =B:FILE ⏎

No Fatal error(s)

A>L80 B:MYCRYPT,B:FILE,B:MYCRYPT/N/E ⏎

MSX.L-80  1.00  01-Apr-85  (c) 1981,1985 Microsoft

Data    0103    072A    < 1575>

41192 Bytes Free
[0104   072A        7]

A>B: ⏎
B>TYPE SAMPLE.DOC ⏎
THIS IS SAMPLE TEXT ·············このテキストの文字コードに12だけ加える

B>MYCRYPT 12 SAMPLE.DOC ⏎···············暗号化

B>TYPE A.OUT ⏎
 TU_,U_,_MY¥XQ, Qd & ··············暗号化されている
B>█                              SAMPLE.DOCのCR, LF, EOF
                                 のコードも暗号化されたため,
                                 改行されていない
```

図5.20 MYCRYPT.MACの実行まで

このプログラムによって暗号化されたファイルは，256 −〈暗号化のオフセット値〉の値で再び暗号化すると，もとのファイルに変換することができます．

なお，このプログラムでは引数のチェックを完全には行っていません．この部分を修正することで，あなただけのより完全な暗号化プログラムを作ってください．

No Fatal error(s)

No Fatal error(s)

43102 Bytes free

図5.20 MYCRYPT.MACの実行結果

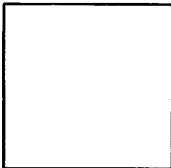
6章

プログラムのデバッグ

―MSX-S BUGを使う―

新しく作ったプログラムには，かならずなんらかの間違いがあるものです．もうご存じだと思いますが，プログラムの間違いは「バグ」(Bug)と呼ばれています．そして，このバグをプログラムのなかから取り除き，プログラムを完全なものにする作業を「デバッグ」(Debug)といいます．これまではとくにデバッグ作業について触れませんでしたが，プログラム作成の一連の作業時間の半分はデバッグに費やされるものだという人もいるくらい大切なものなのです．デバッグのためには，CPU のレジスタ構成やニーモニック，MSX のメモリマップなどさまざまな知識が要求されます．逆にデバッグ作業はこれらの知識を深める絶好の機会であるともいえます．

アセンブリ言語のプログラムのバグのなかでもとくにプログラマの頭を悩ますものは，プログラムの論理的な間違い，つまりアセンブル／リンク・ロードはできるが実行のようすがおかしいというものです．このような論理的なエラーに対しては「デバッガ」(Debugger) というソフトウェア・ツールを用いてデバッグ作業を行うのが一般的です．

本章では，デバッグ作業について考えることにします．そのためにもバグを分類してそれぞれの場合の対処法を説明します．そしてMSX-DOS 用に市販されているデバッガ“MSX-S BUG”を例にとり，実例を見ながらデバッガの働きや機能を考えることにしましょう．

6-1 バグの分類とその対策

ひとくちにデバッグ作業といってもその原因であるバグの種類によって作業の内容はまったく異なったものとなります．まず，ここではバグを大きく2つのパターンに分けてそれぞれについて考えてみることにしましょう．

アセンブリ言語の文法上の誤り

この文法上の誤りはどちらかというと初心者に多いものです．その原因はタイプミスや，勘違いによるもの，あるいはニーモニックの覚え間違いなどによるものでしょう．

これらの間違いは，実は最も簡単に取り除くことができるバグなのです．というのは，アセンブル／リンク・ロードを行うときにアセンブラやリンク・ローダがこれらの誤りを検出してくれるからです．この種のエラーが出たときは，エラーメッセージをたよりに，ソース・プログラムを修正してアセンブルし直せばよいのです．

論理的な誤り

この論理的な誤りというのは，言語の文法的には正しいが期待どおりの動作が行われないというものです．長いプログラムになればなるほど，これらの誤りを完全に取り除くことは難しくなります．

論理的に間違っているといってもいろいろな場合があります．アセンブリ言語に限らずプログラミング言語一般に起こり得るバグの原因としては，アルゴリズムの間違い，プログラムの設計ミスなどが考えられます．ここでは，アセンブリ言語のプログラムに特有のバグとその対策を考えてみましょう．

●ラベルとシンボル

違う場所に同じ名前のラベルを付ける，相対ジャンプできないところに相対ジャンプしようとする，どこにも宣言されていないシンボルを使うなどがこの種のバグの代表的なものです．また，シンボルの長さはいくらでも長くすることはできますが，内部シンボルは16文字まで，外部シンボルは6文字までしか区別されませんから，これによって2つの違ったラベルが同じ名前だと解釈されることもありえます．これらの場合には，アセンブル時あるいはリンク・ロード時に見つかるのであまり問題とはならないでしょう．それよりも誤って本来の飛びさきとは異なった別のラベルへ飛ぶように記述した場合には，見つけるのが大変です．

● PUSH と POP

サブルーチン内でレジスタの値を一時的にスタック領域に保存しておいて，あとでその値を取り出すというのは，よく使われるプログラミング上のテクニックです．この際に，PUSH と POP の順番を間違えるというバグが考えられます．スタック操作では，最後に PUSH したものから順に POP して取り出すようにしなくては正しい値を取り出すことができません．また，PUSH した回数だけ POP しないと正しくもとのルーチンに戻ることすらできなくなります．

また，十分なスタック領域が確保されていないのにスタック操作命令を多用するということも異常な動作の原因となります．スタック操作をするたびにプログラムやワークエリアが破壊されていくことになるからです．これを防ぐには，SP（スタックポインタ）の値をプログラムの先頭で指定しておくことが必要です．

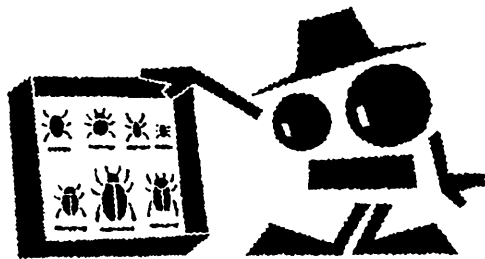
●サブルーチンとのデータの受け渡し

サブルーチンとの間でデータを渡したり，サブルーチンが値を返すようなときに，その受け渡し方は，呼び出す側と呼び出される側で統一しておかなくてはなりません．これは当り前のことですが，実際には仕様変更などの原因でこの種の食い違いを起こしやすいものなのです．たとえば，メインルーチンでは HL レジスタに2バイトの値を入れてサブルーチンを呼んでいる

のに，サブルーチンではDEレジスタの値を参照するというような場合です．このようなことを防ぐためには，きちんと仕様を決めておいて双方でその仕様を守るようにしなくてはなりません．

●サブルーチンやマクロでのデータの破壊

サブルーチンやマクロを使用する際に注意することとして，データの破壊もあげなくてはなりません．このバグは，呼び出す側の親ルーチンと呼び出される側の子ルーチンで同じワークエリアや同じレジスタを違う目的に使用するといった場合に起こります．ワークエリアを共有している場合には，親ルーチンと子ルーチンでそれぞれ別のワークエリアを用意することがその対策となります．レジスタの共有については，親ルーチンが子ルーチンを呼び出すときにあらかじめ必要なレジスタを保存する，あるいは子ルーチンのなかでレジスタ値を保存するなどが対策としてあげられます．また，システムが用意したサブルーチンを呼び出す場合にも同じことがあてはまります．MSX-DOSのシステムコールではすべてのレジスタの内容が保存されているとは限りませんから，必要に応じてレジスタの内容を保存しなくてはなりません．

ここでは代表的なものを取り上げましたが，これ以外にも，フラグの立つ条件を間違ったり，桁あふれに対する考慮がされていない場合，論理演算を勘違いしたり，レジスタを間違うなど，バグとなる要素はいくらでもあります．これらのバグを取り除くためには，自分がCPUになったつもりでソース・プログラムを丁寧に読んでいくか，デバッガのお世話になるしかありません．

6 2 デバッガとその機能

アセンブリ言語のプログラムをデバッグする第一歩は，やはりソース・プログラムを見ることです．しかし，これだけではどこを見てよいのか見当も付かないことがあります．また，フラグを含めたレジスタの値を常に考えながらプログラムを見るのは非常に大変なものです．そこで，このデバッグを効率的に行うために，さまざまな機能を備えたデバッガが必要となるのです．

デバッガは，実行可能なオブジェクト・プログラム(アセンブル／リンクで得られたもの）をメモリに読み込んで，内容を表示したり，プログラムの一部を実行したりして，バグの原因を調べるソフトウェア・ツールです．ここで取り上げるデバッガは，MSX-DOS用に（株）アスキーから発売されている“MSX-S BUG”（以下S-BUGという）です．このS-BUGはただのデバッガではなく，**シンボリック・デバッグ**ができる強力なものです．このシンボリック・デバッグ機能についてはあとで説明します．

デバッガの代表的コマンド

正しく動作しないプログラムをデバッグするためには，まず異常動作の原因であるプログラムの誤りを発見しなくてはなりません．デバッガには誤りを発見するために各種の操作を行うコマンドが備わっています．このコマンドを実行していくことでバグを発見することができるのです．ここでは，S-BUGの持っている代表的なコマンドを見ていくことにしましょう．

・メモリダンプ（D）

任意のメモリ領域の内容を16進数でダンプする

- **メモリ内容の書き換え (S)**

  任意のアドレスのメモリの内容を1バイトごとに表示し，必要があれば変更する

- **メモリデータのブロック転送 (M)**

  任意のメモリ領域を任意のアドレスに転送する

- **メモリ領域の初期化 (F)**

  メモリの任意の領域を任意の1バイトのデータで埋める

- **メモリ領域の検索 (N)**

  任意のメモリ領域から，任意のデータ（列）が含まれている部分を探す

- **各レジスタの内容表示と値のセット (X)**

  CPUのレジスタの内容を表示する．または任意のレジスタの値を変更する

- **ブレークポイント付きのプログラム実行 (G)**

  デバッグ対象プログラムを任意のアドレスから実行する．必要であれば，実行を中止するアドレス（ブレークポイント）を設定する

- **トレース実行 (T)**

  デバッグ対象のプログラムを任意のアドレスから，任意のステップだけ実行するとともに，全ステップでCPUのレジスタの内容を表示する

- **サブルーチンをパスしてトレース (C)**

  基本的にトレース実行と同じだが，CALL命令の場合には，そのCALLで呼ばれるサブルーチンの内部はトレースしないで実行する

・**アセンブル（A）**
入力されたニーモニックをオブジェクトに直して，任意のアドレスから順にメモリに書き込んでいく

・**逆アセンブル（L）**
メモリ上の任意のアドレスのオブジェクト・プログラムから，そのニーモニックを生成して表示する

・**ファイル名の指定（E）**
ファイル名をデフォルトFCB（5CHから）に設定する

・**ディスクリード（R）**
Eコマンドで指定されたファイルを任意のアドレスから読み込む

・**ディスクライト（W）**
任意のメモリ領域の内容を，Eコマンドで指定されたファイル名でディスクに書き込む

・**ヘルプ（？）**
S-BUGのコマンドサマリを表示する

シンボリック・デバッグ

シンボリック・デバッグは，シンボルを用いてデバッグを行うことです．シンボルの値はソース・プログラムでは意味を持っていますが，アセンブル／リンクを行って得られたオブジェクト・プログラムでは，具体的な値やアドレスとなって置き換えられています．デバッグする際にソース・プログラムで用いられているシンボルを使うことができれば，ソース・プログラムとの関係がよくわかるようになり，結局プログラムがデバッグしやすくなるのです．

このS-BUGで用いることのできるシンボルは，外部シンボル（他のモジュールでも使用可能なモジュール間で共通なシンボル）だけです．この理

由は，内部シンボルはモジュールによって異なった値を取ってしまうからです．たとえばあるモジュールが内部シンボル“LOOP”に200Hを割り当てているのに，別のモジュールでは300Hに割り当てるといった具合いです．このとき，最終的なオブジェクト・プログラムではLOOPにはどの値が割り当てられるかを1つに決めることはできません．

シンボリック・デバッグの特徴は逆アセンブル時によく現れます．4章で作成した時計プログラムを例としてとりあげましょう．まず，シンボリック・デバッグを行わない場合，つまり，このプログラムをアセンブル／リンク・ロードして得られた“CLK.COM”のみをS-BUGに読み込ませ，逆アセンブルした場合を見てみます（図6.1）．

```
-L100
0100  C3 06 01     JP    0106
0103  00           NOP
0104  00           NOP
0105  00           NOP
0106  ED 7B 06 00  LD    SP,(0006)
010A  CD B6 01     CALL  01B6 ············
010D  11 22 01     LD    DE,0122         :··アドレスのみが
0110  CD C0 01     CALL  01C0 ············:  表示される
0113  CD C6 01     CALL  01C6 ············
0116  B7           OR    A,A
0117  C2 00 00     JP    NZ,0000
011A  CD 43 01     CALL  0143
011D  CD 55 01     CALL  0155
0120  18 F1        JR    0113
0122  1B           DEC   DE
0123  59           LD    E,C
```

図6.1 シンボルを表示しない逆アセンブル

次に，シンボリック・デバッグを行った場合，つまり，オブジェクトの“CLK.COM”と，シンボル・ファイル（後述）“CLK.SYM”を読み込ませ，逆アセンブルしたものを図6.2に示します．前の図6.1と見比べてください．

```
-L1ØØ
Ø1ØØ    C3 Ø6 Ø1       JP      Ø1Ø6
Ø1Ø3    ØØ             NOP
Ø1Ø4    ØØ             NOP
Ø1Ø5    ØØ             NOP
Ø1Ø6    ED 7B Ø6 ØØ    LD      SP,(ØØØ6)
Ø1ØA    CD B6 Ø1       CALL    Ø1B6 [CLS]
Ø1ØD    11 22 Ø1       LD      DE,Ø122
Ø11Ø    CD CØ Ø1       CALL    Ø1CØ [PUTMSG]
Ø113    CD C6 Ø1       CALL    Ø1C6 [CHKKEY]
Ø116    B7             OR      A,A
Ø117    C2 ØØ ØØ       JP      NZ,ØØØØ
Ø11A    CD 43 Ø1       CALL    Ø143
Ø11D    CD 55 Ø1       CALL    Ø155
Ø12Ø    18 F1          JR      Ø113
Ø122    1B             DEC     DE
Ø123    59             LD      E,C
```


図6.2 シンボルを表示した逆アセンブル

4章でのリストと見比べてもらえばわかりますが，このシンボル付きの逆アセンブルの場合，PUBLIC宣言してあるラベルはその名前が表示されています．

このように，逆アセンブル時には，その値にシンボルが付いて表示されますから，ソース・プログラムとの対比からわかりやすくなるのです．ところで，注意深い人はこの逆アセンブルの結果を見て気付いたかもしれませんが，S-BUGでは，数値の後ろに何も付けなければ16進数と解釈されます．

ここで，10進数で数値を与えたい場合には，数値の後ろにピリオド(.)を，2進数で与えたい場合には（！）を指定します（表6.1）．

入 力	基 数	値(10進)
3	16	3
24	16	36
50.	10	50
6F	16	111
10110!	2	22
10000.	10	10000
2710	16	10000

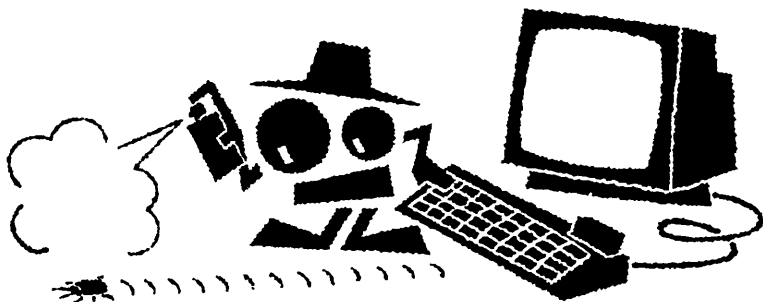
表6.1 S-BUGでの数値の取り扱い

図6.2の例では，シンボル·ファイルをオブジェクト·ファイルと一緒に読み込んでいました．実は，シンボル·ファイルとはプログラムで用いられている外部シンボルの名前と定義された値が収められたファイルなのです．このシンボル·ファイルを参照することで，シンボルと数値を結び付けられるために，シンボリック·デバッグが可能となるのです．このシンボル·ファイル（".SYM"の拡張子が付く）はどのようにして作成するのでしょうか？　シンボル·ファイルの作成はリロケータブル·ファイルからオブジェクトを作るときに一緒に行わせます．具体的には，MSX·L-80にオプションとして"／Y"を与えるというものです．4章の時計プログラム（16進数表示版）のリロケータブル·ファイルからリンク·ロード時に"CLK.COM"と"CLK.SYM"を作成するようすを見てみます．

図6.3 シンボル·ファイルの作成

また，S-BUG のコマンドでもシンボルの表示／定義／解除を実行することができます．それらのコマンドを次に示します．

Y	……	全シンボルと値の表示
Y 〈シンボル〉	……	〈シンボル〉にマッチするシンボルと値の表示
YS 〈シンボル〉	……	新たにシンボルを定義
YX 〈シンボル〉	……	シンボルの消去
YR 〈ファイル名〉	……	シンボル・ファイルを読み込む

シンボリック・デバッグの恩恵は，逆アセンブルのときだけではありません．S-BUG のコマンドのうち，パラメータを取るコマンドでは，数値の代わりにシンボルを用いて指定することもできます．

6.3 S-BUGの実行例

S-BUGを実行する前に，S-BUGでデバッグを行っているときのメモリマップを見ておきましょう．

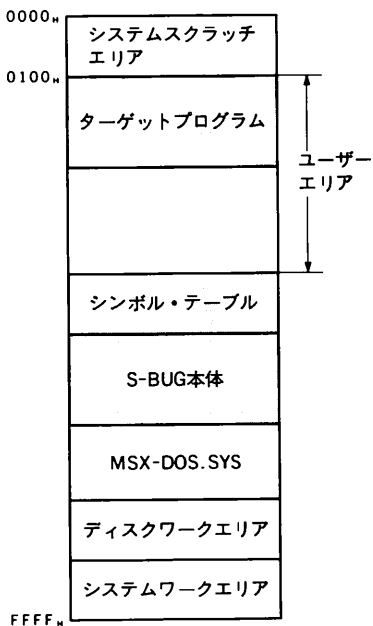



図6.4　S-BUG実行時のメモリマップ

このようにデバッガ自身は，デバッグ対象のプログラムが使うユーザーエリアをできるだけ広く連続的に取れるようにメモリの高位番地のMSX-DOSのシステムのすぐ直前に置かれるのです．

それでは，S-BUG の機能を実際に見てもらうことにしましょう．ここで例として取り上げるのは，さきほどと同じく時計プログラム“CLK.COM”です．

```
-YS HOUR ↵ ··············
HOUR     = 1Ø3 ↵               HOUR, MIN, SECのワークエリアを
-YS MIN ↵ ···············      シンボルとして登録する
MIN      = 1Ø4 ↵
 YS SEC ↵ ···············
SEC      = 1Ø5 ↵
 L145 ↵ ··············145H番地以降を逆アセンブル
Ø145   CD Ø5 ØØ      CALL    ØØØ5
Ø148   7A            LD      A,D
Ø149   32 Ø3 Ø1      LD      (Ø1Ø3 [HOUR]),A
Ø14C   7D            LD      A,L                  --- 新たに登録
Ø14D   32 Ø4 Ø1      LD      (Ø1Ø4 [MIN]),A           したシンボ
Ø15Ø   7C            LD      A,H                      ルがアドレ
Ø151   32 Ø5 Ø1      LD      (Ø1Ø5 [SEC]),A           スと同時に
Ø154   C9            RET                              表示される
Ø155   11 8A Ø1      LD      DE,Ø18A
Ø158   CD CØ Ø1      CALL    Ø1CØ [PUTMSG]
Ø15B   3A Ø5 Ø1      LD      A,(Ø1Ø5 [SEC])------
Ø15E   CD CC Ø1      CALL    Ø1CC [PUTNUM]
Ø161   11 96 Ø1      LD      DE,Ø196
Ø164   CD CØ Ø1      CALL    Ø1CØ [PUTMSG]
Ø167   3A Ø4 Ø1      LD      A,(Ø1Ø4 [MIN])------
Ø16A   CD CC Ø1      CALL    Ø1CC [PUTNUM]
Ø164   CD CØ Ø1      CALL    Ø1CØ [PUTMSG]
Ø167   3A Ø4 Ø1      LD      A,(Ø1Ø4 [MIN])-------
Ø16A   CD CC Ø1      CALL    Ø1CC [PUTNUM]
-X ↵ ··············レジスタを表示する
P _ _0   A =ØØ BC=ØØØØ DE=ØØØØ HL=ØØØØ S=Ø1ØØ P=Ø1ØØ  JP    Ø1Ø6
P _ _0   A'=ØØ B'=ØØØØ D'=ØØØØ H'=ØØØØ X=ØØØØ Y=ØØØØ I=ØØ
-T ↵ ··············100H番地から1ステップトレースする    ---実行の結果PCが106Hになった
P _ _0   A =ØØ BC=ØØØØ DE=ØØØØ HL=ØØØØ S=Ø1ØØ P=Ø1Ø6  LD    SP,(ØØØ6)
P _ _0   A'=ØØ B'=ØØØØ D'=ØØØØ H'=ØØØØ X=ØØØØ Y=ØØØØ I=ØØ
-T ↵ ··············もう1ステップトレースする    ---実行の結果SPの内容が変化した
P _ _0   A =ØØ BC=ØØØØ DE=ØØØØ HL=ØØØØ S=AA29 P=Ø1ØA  CALL Ø1B6 [CLS]
P _ _0   A'=ØØ B'=ØØØØ D'=ØØØØ H'=ØØØØ X=ØØØØ Y=ØØØØ I=ØØ
-T ↵ ··············もう1ステップトレースする           実行の結果PCが
CLS: ··············サブルーチン名が表示される        ---1B6Hになった
P _ _0   A =ØØ BC=ØØØØ DE=ØØØØ HL=ØØØØ S=AA27 P=Ø1B6  LD    DE,Ø1BD
P _ _0   A'=ØØ B'=ØØØØ D'=ØØØØ H'=ØØØØ X=ØØØØ Y=ØØØØ I=ØØ
-^C ↵ ··············CTRL+C を押して終了する
A>█
```

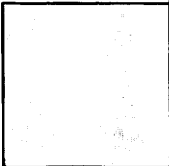
図 6.5　S-BUG の機能の実行例

7章

応用プログラミング

―MSX固有の機能を引き出す―

これまでは，MSX-DOSの機能のみを使ってプログラムを作ってきました．つまり，MSX-DOS上で動くプログラム(COM形式のファイル）をMSX-DOSのソフトウェア・ツールを用いて作成したいたのです．また，そのプログラムから周辺機器（キーボード／ディスプレイ／ディスクなど）を用いるときにはシステムコールを利用していました．しかし，MSX-DOSはMSX上のディスク・オペレーティング・システムであり，私たちは最終的にはMSXで動くプログラムの作成を目的としているのですから，かならずしもDOSのプログラムという形にとらわれる必要はないのです．

本章では，MSX-DOSでのプログラム開発の応用について2つの面から考えていきます．最初にもう1つのMSXのプログラミング環境といえるBASICに注目し，BASICのプログラムとMSX-DOS上で開発したアセンブリ言語のルーチンを組み合わせることで1つのまとまったプログラムを作ります．そしてもう1つのアプローチとして，MSX-DOS上で動くプログラム(COM形式のファイル)を，MSXの機能を活かしたものにすることを考えます．このためにはさまざまな周辺機器を扱うためにBIOSと呼ばれるルーチン群を呼び出すことが必要となりますので，この呼び出し方も含めて説明していくことにしましょう．

7 1 BASICとの組合せ

BASICといえばみなさんもいろいろなイメージを持っているかもしれませんが，気軽にプログラミングできるところは誰もが認める長所といえるでしょう．なお，ここでいっているBASICとは単にBASIC言語を指しているのではなく，BASIC言語のプログラムの作成／実行／デバッグというさまざまな作業を行う場所，つまり**「プログラミング環境」**としての意味を含めたものです．

しかし，BASICのプログラム実行速度はあまり速くありません．この理由はその実行方式や，実行時にエラーチェックをしているなどの理由によるものですが，これではなかなか実用的なプログラムを作ることができません．BASIC言語のプログラムも書き方によってある程度高速化できるのですが，それには限界があります．

そこでBASICのプログラムとマシン語のプログラムを組み合わせるという考えが出てくるのです．一般的には，BASICとマシン語のプログラムの機能分担は表7.1のように決めます．

	特　徴	機能分担
BASIC	・プログラミングしやすい ・命令が豊富なため，高度な機能の利用が容易	・スピードをとくに要求されないメインルーチン等
マシン語	・実行速度が速い ・ハードウェアに密着した記述が可能	・スピードを要求される一部のサブルーチン

表7.1　BASICとマシン語の特徴と機能分担

BASIC 環境でのマシン語

MSX-DOS の外部コマンド（COM 形式のファイル）では，そのファイルが 100H 番地から読み込まれて，そこから実行されるのですが，BASIC 環境ではマシン語はどのように読み込まれ実行されるのでしょうか？

●マシン語プログラムを入れるメモリ領域と保護

BASIC プログラムを実行しているときのメモリマップを図 7.1 に示します．

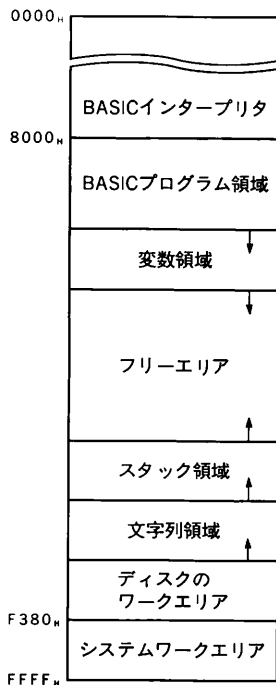



図 7.1　BASIC プログラム実行時のメモリマップ

このときにマシン語プログラムをどこに配置すればよいのでしょうか？もしマシン語のプログラムを入れた領域にBASICの変数領域が重なると，マシン語のプログラムが破壊されてしまいます．ですから，マシン語領域は，他の目的で使われないように確保しなければなりません．この領域の確保は，BASICのCLEAR命令で行います．CLEAR命令は，変数の初期化，文字列領域の大きさの指定をする命令ですが，ほかにも変数の領域の上限（領域の最高位アドレス）を指定することができます．さらにBASICで用いるメモリの上限を指定することができるので，マシン語のプログラムはそのアドレスより上で，システムワークエリアより下を自由に使うことができます（図7.2）．

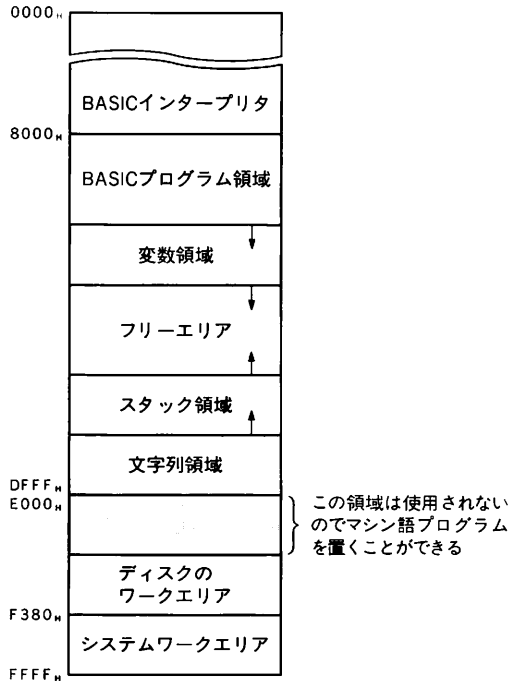



図7.2 CLEAR命令でマシン語プログラムの領域を確保

では，システムワークエリアの下限はどこなのでしょうか？ ディスクの付いていないMSXでは，ワークエリアはF380H番地以降になっています．しかし，ディスクが付いている場合には，さらにディスクアクセスのためのワークエリアが確保されますから，システム全体のワークエリアの下限はもっと低くなります．現在市販されているディスク装置で最も多くワークエリアを必要とするのは2DDドライブのものですが，このワークエリアについても製品によって微妙に異なっています．これに対処するためにユーザーが使うメモリの上限はDE3FH番地までにすることが推奨されています．

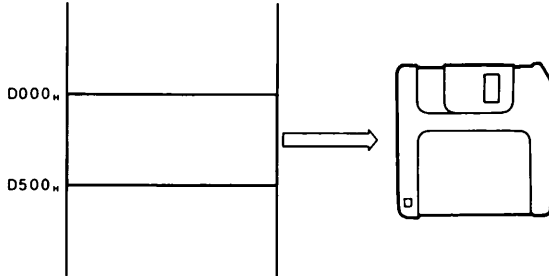
しかし，DE3FH番地まで全部を使う必要はまったくありません．B800H番地から，あるいはD000H番地からのようにきりのよい数字からマシン語ルーチンを置くのが一般的です．もっとも，AB91H（エビ食い）番地からというように語呂合わせで決めてもいっこうにかまわないのですが．

●マシン語ファイル

マシン語のプログラムを置くということは，メモリにマシン語のデータをセットすることです．このためには，BASICのPOKE命令（メモリの指定された番地にデータを書き込む命令）を用いることでも実現できます．この方法は，BASICのなかで短いマシン語のルーチンを用いるような場合には便利なのですが，マシン語のプログラムが大きくなるにつれて現実的ではなくなります．こうなってくるとマシン語プログラム全体を1つのファイルとしてまとめて読み込んだり，書き込んだりする命令を用いる方が簡単です．メモリの内容をファイルにするのがBSAVE命令で，マシン語のファイルをメモリに読み込むのがBLOAD命令です．

BSAVE命令では，ファイルに保存する領域の先頭アドレスと，終了アドレスをパラメータとして受け取りますが，これらの情報は，そのままファイルの先頭に付加されます．BLOADするときには，ファイルに記録された情報から読み込むアドレスを決定するのです（図7.3）．

なお，先頭と終了のアドレスをファイルに付加しているといいましたが，具体的なファイルの記録フォーマットは図7.4のようになっています．

BSAVE "SAMPLE.BIN", &HD000, &HD500

BLOAD "SAMPLE.BIN"

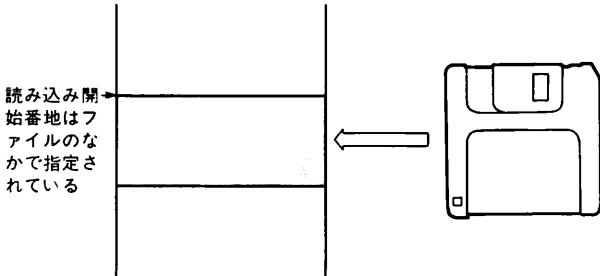



図 7.3 BSAVE 命令と BLOAD 命令

BSAVE "FILE.BIN", &H8E71, &H977A, &H93EC

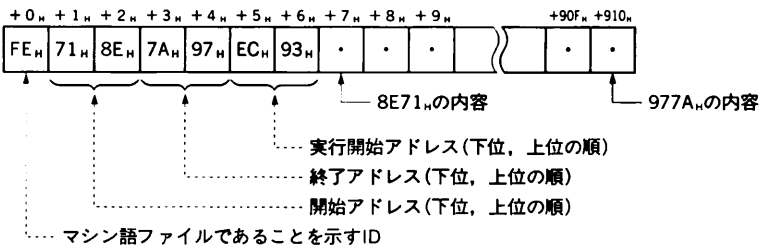



図 7.4 マシン語ファイルのフォーマット

● USR 関数

メモリ上に置かれたマシン語のルーチンを BASIC から呼び出すためには，まず次のようにマシン語の呼び出しアドレスを宣言します．

```
DEFUSR = &HD000
```

この宣言で，マシン語のプログラムが D000H 番地から始まることが宣言できたので，次のようにして呼び出してやればよいのです．

```
X = USR (0)
```

この命令ではマシン語のサブルーチンを呼び出しているので，マシン語の RET 命令で戻ってきます．この呼び出しさきは USR0 から USR9 までの 10 種類持つことができます．USR だけで後ろに数字を付けない場合は USR0 であると解釈されます．このマシン語呼び出しが関数の形になっているのは，BASIC からマシン語ルーチンへマシン語ルーチンから BASIC へと値の受け渡しができるようにするためなのです．

この値の受け渡しについてですが，マシン語ルーチンから BASIC に値を渡すのは難しいので本書では扱いません．BASIC からマシン語ルーチンに値を渡す方法については説明しておきましょう．マシン語のルーチン側では，呼び出されたときの A レジスタの値で引数の型を知ることができます．その型による値の渡し方は図 7.5 のようになっています．

● 周辺機器の入出力

MSX-DOS では周辺機器との入出力はシステムコールという形で統一的に行われましたが，BASIC 環境ではどのようにして行うのでしょうか？入出力のためには BIOS (Basic Input/Output Subroutines) と呼ばれるルーチン群が用意されています．この BIOS は BASIC インタプリタの ROM のなかにはいっていて，そのルーチンを呼び出すことで周辺装置との入出力を行います．BIOS のルーチンの呼び出し番地とその機能については代表的なものを巻末の Appendix に載せてありますのでそちらを参照してください．

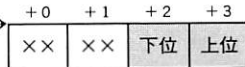
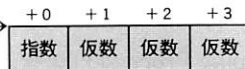
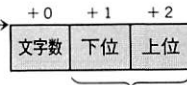



図 7.5 引数の型と値の渡し方

また，BIOSにはディスクの入出力が含まれていません．ただし，MSXにディスク装置が付いているときにはMSX-DOSでの入出力と同様のシステムコールをBASIC環境でも利用できます．BASIC環境でのシステムコールとMSX-DOSでのシステムコールの違いは，呼び出し番地が異なることです．

```
MSX-DOS  ……  0005H
BASIC    ……  F37DH
```

マシン語サブルーチンの作成

BASICでのマシン語のプログラム（サブルーチン）の配置や，実行の仕方などを見たところで，実際にマシン語のプログラムをMSX-DOS上で開発するよううすを見ていくことにしましょう．サンプル・プログラムは，マシン語ルーチンの高速性を示すために画面を直接扱うという例を取り上げます．

画面に文字を表示するにはVRAM(ビデオRAM)にデータを書き込む必要があります．このVRAMはメインメモリ上にあるのではなく，画面表示用のLSIであるVDP (Video Display Processer)の管理下に置かれていて，CPUからはI/Oポートを経由してアクセスしなければなりません（図7.6）．

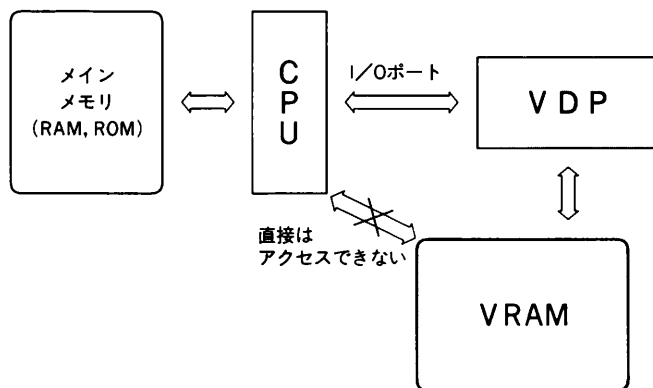



図7.6　VDPとVRAM

MSXには用途に応じてさまざまな画面モードが用意されていますが，ここではSCREEN 1モードを使うことにします．このモードは基本的には横32×縦24文字の表示ができるテキスト画面ですが，スプライト機能を使えるので簡単なアニメーションには便利なのです．このSCREEN 1モードでのVRAMの構成を図7.7に示します．

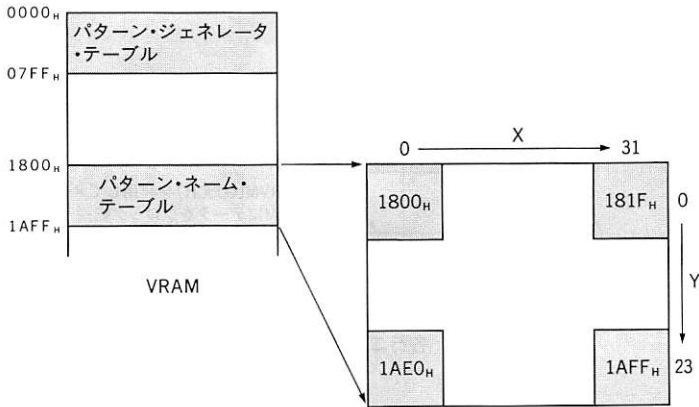



図 7.7 SCREEN 1 モード

パターン・ジェネレータ・テーブルは，それぞれの文字のフォント（形）を決めるデータが収納されています．また，パターン・ネーム・テーブルには，画面に表示される文字が，それぞれの座標に対応したアドレスに文字コードの形で収納されています．

この画面を扱うマシン語サブルーチンを作るのですが，ここでは次に示す 2 つのマシン語サブルーチンを作ります．

・USR0 …… 画面の右端に背景を描きます．具体的には，受け取った値を Y 座標として，その位置より上を空白で埋め，下側を“#”で埋めるというものです．このルーチンでは WRTVRM（004DH）という VRAM にデータを 1 バイト書き込む BIOS を使用します

・USR1 …… 画面全体を左に 1 文字分スクロールさせます．具体的には各行ごとに図 7.8 のようにしてスクロールを行います

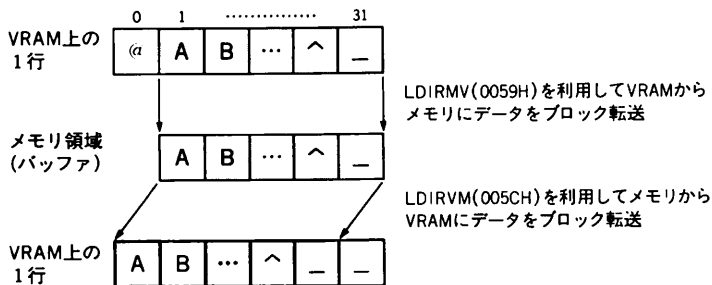



図 7.8 各行ごとの横スクロール

では，この2つのルーチンを含んだマシン語プログラムをリスト 7.1 に示します．

```
; USRFUNC.MAC : BASIC'S USR FUNCTION SAMPLE

        .Z8Ø
                                SCREENモード1でのパターン・
VRAMAD  EQU     18ØØH.........  ネーム・テーブルのアドレス
WIDTH   EQU     32 ............ 画面の桁数(SCREENモード1)
LINES   EQU     24 ............ 画面の行数(SCREENモード1)

        CSEG ............ 相対アドレス宣言(本文参照)

USRØ:   JP      PUTPAT
USR1:   JP      SCROLL

;--- Put Background Pattern --- ............ 背景パターンの表示
PUTPAT:                                      (画面の右端)
        CP      2       } 引数が整数でなければ戻る
        RET     NZ      }
        INC     HL      }
        INC     HL      } B＝引数の下位8ビット(境界のY座標)
        LD      B,(HL)  }
        PUSH    BC
        LD      HL,VRAMAD+WIDTH-1 }
        LD      C,' '             } 画面の上から境界まで
        CALL    PUTSUB            } を空白で埋める
        POP     BC
        LD      A,LINES }
        SUB     B       }
```

```
        LD      B,A         境界から画面の下まで
        LD      C,'#'       を"#"で埋める
        CALL    PUTSUB
        RET

;--- Put Pattern Subroutine --- ……アドレスHLから縦に文字C
PUTSUB:                           をB個表示する
        INC     B
        DEC     B       Bが0ならば戻る
        RET     Z
PUTS1:
        LD      A,C
        CALL    004DH ……VRAMに1バイト書き込む(BIOS)
        LD      DE,WIDTH    次の行のアドレスを求める
        ADD     HL,DE
        DJNZ    PUTS1
        RET

;--- Scroll Left (SCREEN 1) --- ……画面を左に1文字分スクロールさせる
SCROLL:
        LD      B,LINES ……B＝スクロールさせる行数
        LD      HL,VRAMAD ……HL＝スクロール開始行のVRAMの
SCROL1:                      先頭アドレス
        PUSH    HL
        PUSH    BC
        CALL    SCRLIN ……1行を左にスクロールする
        POP     BC
        POP     HL
        LD      DE,WIDTH    次の行のアドレスを求める
        ADD     HL,DE
        DJNZ    SCROL1
        RET

;--- Scroll One Line --- ……1行を左に1文字分スク
;[IN]   HL = VRAM Line Address    ロールさせる
SCRLIN:
        PUSH    HL
        INC     HL              行の2文字目から31文
        LD      DE,SCRBUF       字をSCRBUFに
        LD      BC,WIDTH-1
        CALL    0059H ……VRAMからメモリ上に転送(BIOS)
        POP     DE              SCRBUFから31文字を行の
        LD      HL,SCRBUF       1文字目以降に
        LD      BC,WIDTH-1
        CALL    005CH ……メモリからVRAMにデータを
        RET                 転送(BIOS)
SCRBUF: ……スクロールする画面(1行)の一時退避領域

        END
```

リスト7.1 USRFUNC.MAC

リストのなかで注意しておいてほしいのは，絶対モードの指定をするASEGを用いているのではなく，相対モードの指定をするCSEGを用いていることです．さて，このソース・プログラムのアセンブル／リンクを行うのですが，単なるサブルーチンなので，これまでのようにCOM形式のファイルを作っても実行することができません．ここではあとでマシン語のファイルを作る都合で**「インテルHEX形式」**と呼ばれるオブジェクト・ファイルを作ります．このインテルHEX形式のオブジェクト・ファイルは，リンク・ローダL-80に／Nオプションを付けると同時に，／Xオプションを付けることで作成されます．L-80には，／P：D000と指定してオブジェクトをロードするアドレス（この場合D000H番地）を指定します．このようにオブジェクトをロードするアドレスを変えられるようにCSEGを用いていたのです．

```
A>M8Ø =B:USRFUNC ⏎

No Fatal error(s)

A>L8Ø /P:DØØØ,B:USRFUNC,B:USRFUNC/N/X/E ⏎
        └----D000番地以降にロードする  └------HEXファイルの出力を指定
MSX.L-8Ø  1.ØØ  Ø1-Apr-85  (c) 1981,1985 Microsoft

Data    DØØØ    DØ57    <   87>

42723 Bytes Free
[ØØØØ   DØ57      2Ø8]

A>TYPE B:USRFUNC.HEX ⏎ ···············HEXファイルを画面に表示させる
:2ØDØØØØØC3Ø6DØC32EDØFEØ2CØ232346C5211F18ØE2ØCD2ØDØC13E189Ø47ØE23CD2ØDØC9BD
:2ØDØ2ØØØØ4Ø5C879CD4DØØ112ØØØ191ØF6C9Ø618210Ø18E5C5CD41DØC1E1112ØØØ191ØF3A5
:17DØ4ØØØC9E5231157DØØ11FØØCD59ØØD12157DØØ11FØØCD5CØØC95F
:ØØØØØØØ1FF

A>█
```

図7.9 USRFUNC.MACのアセンブル／リンク・ロードのようす

これでマシン語サブルーチンのインテルHEXファイルが得られました．このファイルをBLOADできるマシン語のファイル（バイナリ・ファイル）にする変換は，MSX-DOS TOOLSに含まれるBSAVEコマンドを使うことで実現できます．その実行の書式は次のとおりです．

BSAVE　<HEX形式ファイル名>　[<バイナリ・ファイル名>]

このBSAVEコマンドを使ってバイナリ・ファイルを作るようすを図7.10に示します．

```
A>BSAVE B:USRFUNC.HEX B:USRFUNC.BIN ⏎ ··············BSAVEコマンドの実行

A>DIR B:*.BIN ⏎
USRFUNC  BIN      94 88-04-14 10:57p
        1 file   680960 bytes free
A>█
```

図7.10　バイナリ・ファイルの作成

このようにMSX-DOS TOOLSのBSAVEコマンドを使えば，インテルHEXフォーマットのファイルをバイナリ・ファイルに変換することができますが，この処理の内容はそんなに難しいものではありませんので，BASICで作った簡単な変換プログラムをリスト7.2に載せておきましょう．

```
100 '***** HEX2BIN.BAS *****
110 CLEAR 500, &HA000
120 INPUT "Hex File Name "; F$
130 OPEN F$ FOR INPUT AS #1
140 GOSUB 200: TA=A
150 GOSUB 200: IF NOT(EOF(1)) THEN 150
160 INPUT "Bin File Name "; F$
170 BSAVE F$, TA, EA
180 PRINT "END": END
```

```
190 '--- Input Line ---
200 LINE INPUT #1,X$
210 IF LEFT$(X$,1) <> ":" THEN 320
220 N=VAL("&H"+MID$(X$, 2, 2))
230 IF N=0 THEN RETURN
240 A=VAL("&H"+MID$(X$, 4, 4))
250 FOR I=1 TO N
260   D=VAL("&H"+MID$(X$, I*2+8, 2))
270   POKE A+I-1,D
280 NEXT I
290 EA=A+N-1
300 RETURN
310 '--- Illegal Data ---
320 PRINT "ILLEGAL DATA": END
```

リスト 7.2　HEX2BIN.BAS

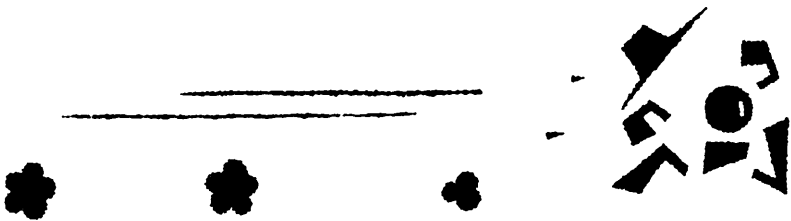
これでマシン語のサブルーチンとなるファイル (USRFUNC.BIN) ができあがりました．このプログラムを呼び出す BASIC のプログラムの例をリスト 7.3 に示します．

```
100 '*** BASIC Main Program ***
110 CLEAR 100,&HCFFF: DEFINT A-Z
120 SCREEN 1,3: COLOR 3,4,0: CLS
130 FL$="B:USRFUNC.BIN"} マシン語ファイルの読み込み
140 BLOAD FL$          }
150 DEFUSR0=&HD000: DEFUSR1=&HD003 ……………USR関係の使用宣言
160 GOSUB 520
170 Y=20: DY=1: SW=1: TI=1
180 '--- Main Loop ---
190 IF RND(1) < .5 THEN DY=-DY
200 Y=Y+DY
210 IF Y > 23 THEN Y=23
220 IF Y < 15 THEN Y=15
230   D=USR1(0): D=USR0(Y) ……………スクロールと背景の表示(マシン語)
240 ' GOSUB 420: GOSUB 330 ……………BASICの場合
250 TI=TI-1
260 IF TI = 0 THEN TI=2: GOSUB 280
270 GOTO 180
280 '--- Put Bird ---
290 IF SW THEN SW=0 ELSE SW=1
300 PUT SPRITE 0, (60, 30), 10, SW
310 PUT SPRITE 1, (60, 30), 1,  SW+2
320 RETURN
```

```
330 '--- Put Back Pattern by BASIC ---
340 AD = &H1800+31
350 FOR I=0 TO Y-1
360   VPOKE  AD,ASC(" "): AD=AD+32
370 NEXT I
380 FOR I=Y-1 TO 23
390   VPOKE  AD,ASC("#"): AD=AD+32
400 NEXT I
410 RETURN
420 '--- Scroll by BASIC ---
430 LA=&H1800
440 FOR I=0 TO 23
450   AD=LA
460   FOR J=0 TO 30
470     VPOKE AD,VPEEK(AD+1): AD=AD+1
480   NEXT J
490   LA=LA+32
500 NEXT I
510 RETURN
520 '--- Sprite Initialize ---
530 RESTORE 620
540 FOR I=0 TO 3
550   A$=""
560   FOR J=0 TO 31
570     READ B$: A$=A$+CHR$(VAL("&h"+B$))
580   NEXT J
590   SPRITE$(I)=A$
600 NEXT I
610 RETURN
620 '--- Sprite Data ---
630 DATA 00,00,00,09,0D,0D,45,F5
640 DATA FD,3D,0E,03,00,00,00,00
650 DATA 00,00,00,00,00,80,DC,D7
660 DATA DC,F8,F0,E0,00,00,00,00
670 '
680 DATA 00,00,00,00,00,00,00,07
690 DATA 1F,7F,EF,4F,03,00,00,00
700 DATA 00,00,00,00,00,00,1C,B7
710 DATA FC,70,A0,C0,E0,C0,00,00
720 '
730 DATA 00,00,00,12,12,12,0A,0A
740 DATA 02,C2,31,0C,03,00,00,00
750 DATA 00,00,00,00,00,00,00,28
760 DATA 22,04,08,10,E0,00,00,00
770 '
780 DATA 00,00,00,00,00,00,00,00
790 DATA 00,00,10,A0,4C,03,00,00
800 DATA 00,00,00,00,00,00,00,08
810 DATA 02,8C,50,20,10,20,C0,00
```

リスト 7.3　BIRD.BAS

ちなみに，このプログラムではマシン語のルーチンの処理とほとんど同じ内容をBASICでも記述してあります．マシン語のルーチンの代わりにBASICのルーチンを呼ぶためには，230行をREM文に変え，240行のシングル・クォーテーション (') を取ってください．BASICルーチンを呼ぶ場合の実行では，マシン語の場合との実行の速度の差が実感できるでしょう．

このようにBASIC上から利用できるマシン語のプログラムもMSX-DOSで開発できることがわかりました．MSX-DOSのコマンドを作る場合に比べて多少手順が複雑になっています．しかし，プログラムの大半をBASICで組み，速度に関係する部分をマシン語で書くようにすれば全体としての開発効率は向上するでしょう．

7 2 DOSからBIOSを呼び出す

これまで見てきたのはBASICにマシン語ルーチンを組み込むという例でした．BASIC環境では，実行する速度が問題とならない場所はBASICのプログラムに担当させることができますし，データの入出力のためのBIOSルーチンも揃っているので，マシン語ルーチンの負担は少なくてすみました．ただ，Z80 CPUのメモリ空間64Kバイトのうち，0H～7FFFHまでの32KバイトはBASICのROMが占有するため，マシン語プログラム領域としてせいぜい20Kバイト程度しか利用できないという問題もあります．

いっぽう，MSX-DOS上の外部コマンドとなるプログラムではどうでしょうか？　外部コマンドが使うことのできるメモリ領域はもちろん“TPA”です．TPAの大きさは，機種やドライブ構成によって多少変化しますが，およそ50Kバイトあります．つまり，マシン語プログラムで自由に使える領域の大きさでは，圧倒的にMSX-DOS環境の方が優っているのです．そこで，広いメモリ領域を活用するために，MSX-DOS環境でのプログラミングの応用を取り上げることにしましょう．

MSX-DOS環境で用意されているシステムコールは，VDPやPSGの操作やジョイスティックの読み取りなどの処理を行うことができません．いろいろな周辺機器を活かすためには，MSX-DOS環境においてもBIOSを呼び出すことが必要となります．しかし，前にも述べたように，BIOSはBASICインタプリタのROMの内部にあります．ですから，MSX-DOSが走っている状態では，図7.11のようにRAMの裏側に隠れていて簡単に呼び出すことができません．

MSXシステムには，メモリを管理するために「**インタースロットコール**」が用意されています．MSX-DOS環境から隠れているBIOSルーチンを呼び出すためには，このインタースロットコールを使えばよいのです．

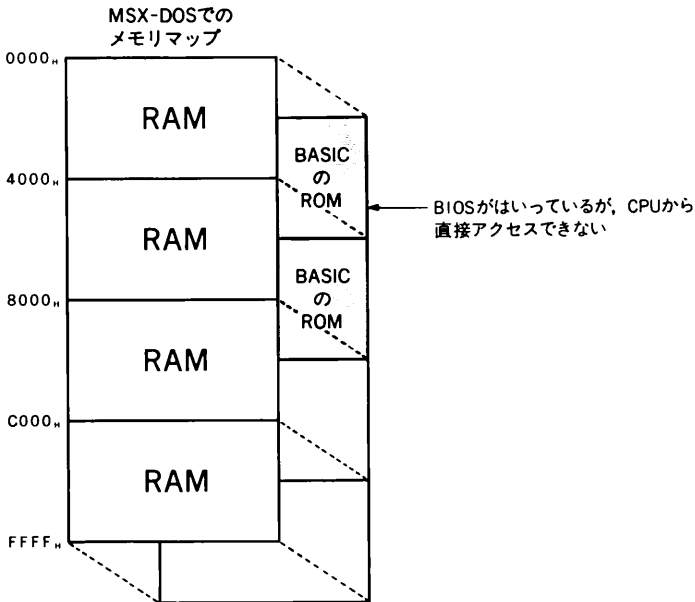



図 7.11 MSX-DOS でのメモリマップと BIOS

インタースロットコール（スロット間コール）

●スロットとページ

Z80 が直接アクセスできるメモリ空間は 64K バイトです．しかし MSX では，この 64K バイトという制限を超えて，より大きなメモリ空間を管理できるようになっています．この大きなメモリ空間は，「スロット」と呼ばれる考え方で，同一のアドレスに複数のメモリを割り当てることにより実現しています（図 7.12）．

各スロットは 64K バイトのメモリ空間を持っていますが，それを 16K バイトごとに「ページ」として 4 つに分けて管理しています．CPU はページごとに任意のスロットを選択してメモリにアクセスできるようになっています（図 7.13）．

図 7.12　スロット

スロット0　スロット1　スロット2　スロット3

0000H
ページ0
4000H
ページ1
8000H
ページ2
C000H
ページ3
FFFFH

A B C D
E F G H
I J K L
M N O P

CPUから見た
メモリマップ
A J G H

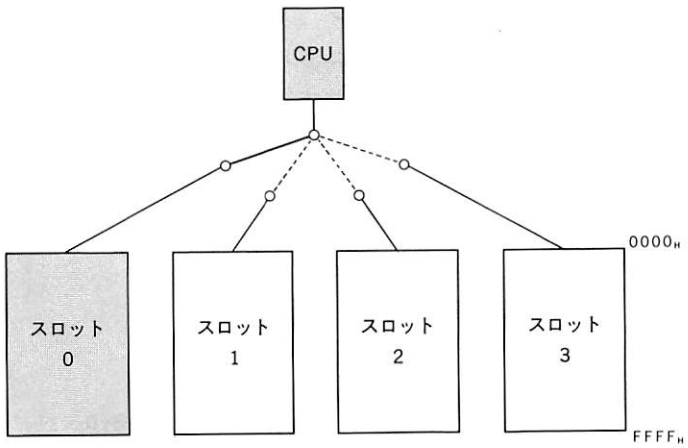
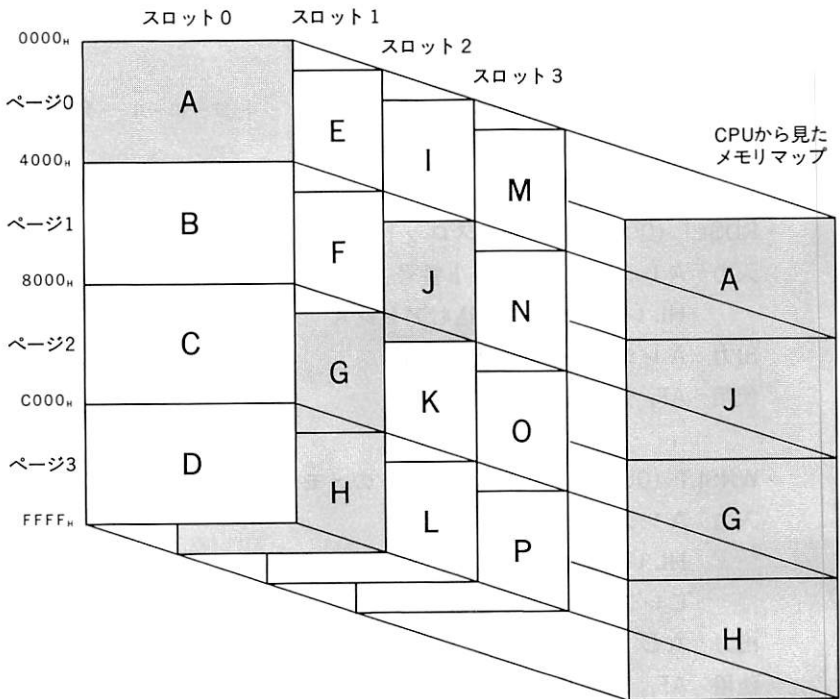



図 7.13　ページ選択の例

●インタースロットコール・ルーチン

BIOSやMSX-DOSのシステムには，各スロット間で相互に呼び出しができるようなルーチン群が用意されています．このスロット間の呼び出しのことをインタースロットコールといい，そのためのルーチン群をインタースロットコール・ルーチンといいます．これらのルーチンで用いられるスロット番号は図7.14のようになっています．

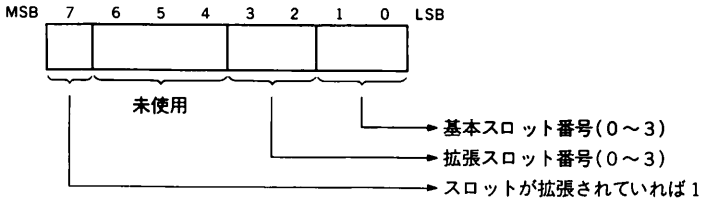



図7.14 スロット番号

MSX-DOSでサポートされているインタースロットコール・ルーチンは次のものです．

・RDSLT (000CH)　他のスロットのメモリの内容を読む

入力　Aレジスタにスロット番号
　　　HLレジスタに読み込むアドレス
出力　Aレジスタに読み出した値
使用　AF，BC，DE

・WRSLT (0014H)　他のスロットのメモリにデータを書き込む

入力　Aレジスタにスロット番号
　　　HLレジスタに書き込むアドレス
　　　Eレジスタに書き込む値
出力　なし
使用　AF，BC，D

・CALSLT（001CH）　他のスロットのサブルーチンを呼び出す

入力　IYレジスタの上位8ビットにスロット番号
　　　IXレジスタに呼び出すルーチンのアドレス
出力　呼び出しさきのルーチンによって決まる
使用　呼び出しさきのルーチンによって決まる

・CALLF（0030H）　他のスロットのサブルーチンを呼び出す

入力　このルーチンをコールする命令の直後にスロット番号（1バイト），アドレス（2バイト）を置く
出力　呼び出しさきのルーチンによって決まる
使用　呼び出しさきのルーチンによって決まる

・ENASLT（0024H）　スロットをページ単位で切り換える

入力　Aレジスタにスロット番号
　　　HLレジスタの上位2ビットに切り換えるページ
出力　なし
使用　すべてのレジスタ

● BIOSの呼び出し方

BIOSを呼び出すためには，BIOSのはいっているROMのスロット番号と呼び出すアドレスの値がわかっていなければなりません．BIOSのはいっているROMのスロット番号は，MSXシステム・ワークエリアのなかのEXPTBL(FCC1H)に設定されています．ですから，呼び出すためには次のようにすればよいのです．

```
LD     IX,〈呼び出すアドレス〉
LD     IY,(EXPTBL-1)
CALL   CALSLT
```

ここでは，IYレジスタにEXPTBL−1の内容を入れることで，IYレジスタの上位8ビットにEXPTBLの内容を代入しています．

LESSコマンドを作る

● LESSコマンドの由来

では，最後に実用的なプログラムを作ることにしましょう．テキスト・ファイルをちょっと読むといった目的には，DOSの内部コマンドのTYPEや，MSX-DOS TOOLSにも含まれるMOREコマンドなどがよく用いられます．しかし，これらのコマンドではいったん画面からスクロールして消えた部分を読もうとすると，プログラムを終了したあとで再び同じコマンドを実行するはめになります．そんなめんどうなことをしなくても，少し前に戻ってテキストを見られるようなツールがあれば便利でしょう．このような目的で作られた逆戻り可能なファイル表示プログラムが“LESS”です．このLESSという言葉はMOREの反対という意味から名付けられたもので，そのオリジナルはUNIXというOS上のツールとして流布していたものです．UNIX版の“less”にはいろいろな機能が含まれているのですが，ここでは機能を絞り込んで**表7.2**のような最低限のコマンドだけを用意した超簡易版を作ることにします．

コマンド	機　能
[H]，[h]	ファイルの先頭に移動
[T]，[t]	ファイルの末尾に移動
[J]，[j] [↵]	1行進む
[K]，[k]	1行戻る
[F]，[f] [SPACE]	1ページ(23行)進む
[B]，[b]	1ページ(23行)戻る
[Q]，[q] [CTRL]＋[C]	終　了

表7.2　LESSのコマンド一覧

● LESSの書式

LESSはファイルを表示するコマンドですから，実行時にはファイル名を指定しなくてはなりません．そこで書式は次のように決めます．

LESS 〈ファイル名〉

●バッファの使用

このプログラムでは，ファイルを頭から順に読む場合だけでなく，もとに戻って読むこともできます．そこで，ファイル全部をあらかじめメモリ上のバッファに読み込んでおく方法を用いると簡単になります．またファイルを読み込んでバッファに書き込むときに，VRAMに転送するのに都合のよい形にデータを展開しておくようにして，画面表示の際の負担を減らすようにします．

●画面モード

このプログラムではテキストを表示するのですから，できるだけ多くの文字が表示できる方が有利です．そうすると画面モードはSCREEN 0モードとなります．SCREEN 0モードはMSX2専用の横80文字モードとMSX／MSX2のどちらでも使える横40文字モードがあります．それぞれのモードでのパターン・ネーム・テーブルは図7.15のような構造になっています．

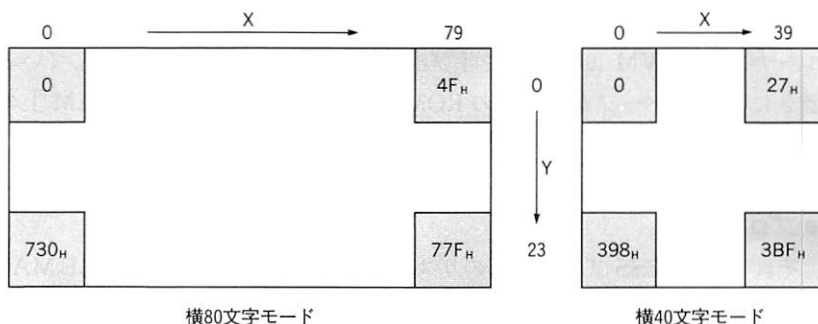



図7.15 SCREEN 0モードのパターン・ネーム・テーブル

●モジュール構成

このプログラムは，超簡易版であるとはいえ，かなりの量のプログラムとなります．そこで考えられるのがモジュール分割です．ここで分割したモジュールごとの役割は次のようになっています．

LESS.MAC …… メインルーチン／コマンド入力／コマンドの処理
BUF.MAC …… バッファ領域への読み込み／バッファ領域の表示
SCR.MAC …… 画面の初期化／VRAM へのデータ転送
FILE.MAC …… ファイルの読み込み

● SCR.MAC でのインタースロットコール

画面を扱うモジュール SCR.MAC ではいくつかのインタースロットコールを行います．

画面の初期化ルーチンでは，画面モードを設定する CHGMOD (005FH) と VRAM をデータで埋める FILVRM (0056H) の 2 つの BIOS を呼ぶためにインタースロットコール (呼び出し) を使います．また，VDP に書き込むときの I／O ポート番号は，BIOS のある ROM の 0007H 番地に記録されているので，この値を取り出すためにインタースロットコール（読み出し）を行います．

VRAM へのデータ転送ルーチンでは，VRAM 書き込みアドレス設定の BIOS である SETWRT (0053H) を呼び出し，データを I／O ポートをとおして VDP に転送するという方法をとっています．BIOS にあるデータ転送ルーチン LDIRVM (005CH) を呼び出さないのは，BIOS を呼び出しているときにはそのページが BIOS の ROM に切り換わってしまい，RAM 上のデータの転送ができないからです．

●プログラム

それでは，LESS プログラムのリストを示すことにしますが，FILE.MAC は 5 章で作成したものを流用しますので，そちらを参照してください．また，MSX を使っている場合には横 80 文字モードが使えませんから，LESS.MAC の冒頭でのシンボル WIDTH の定義を 80 から 40 に変更してください．

```
;***** LESS.MAC *****

        EXTRN   BUFTOP…………外部で定義されたラベル
        EXTRN   TXTDSP, TXTEND, BUFEND…………外部で定義され
        EXTRN   INIT, FOPENR   ｝外部で定義された  たワークエリア
        EXTRN   SETBUF, PUTBUF ｝サブルーチン

        PUBLIC  WIDTH, LINES…………外部から参照されるシンボル

        .Z8Ø

WIDTH   EQU     8Ø…………画面の横幅(MSX2の場合, MSXでは40変更する)
LINES   EQU     23…………画面の行数－1

BOOT    EQU     ØØØØH
SYSTEM  EQU     ØØØ5H
FCB1    EQU     ØØ5CH…………デフォルトFCB の1つ目
DTA     EQU     ØØ8ØH…………デフォルトDTA
CR      EQU     ØDH  ｝
LF      EQU     ØAH  ｝文字コードの定義
EOS     EQU     '$'  ｝
ESC     EQU     1BH  ｝

SYSCAL  MACRO   CALNO…………システムコールのマクロ
        LD      C,CALNO
        CALL    SYSTEM
        ENDM

PUTMSG  MACRO   MSGPTR…………メッセージ表示マクロ
        LD      DE,MSGPTR
        SYSCAL  Ø9H
        ENDM

ERREXT  MACRO   ERRMSG…………エラーメッセージを表示し
        PUTMSG  ERRMSG      ブートするマクロ
        JP      BOOT
        ENDM

CPHLDE  MACRO…………HLレジスタとDEレジスタを
        PUSH    HL     比較するマクロ
        OR      A
        SBC     HL,DE
        POP     HL
        ENDM

        CSEG
```

```
;--- Main Routine ---
MAIN:
        LD      HL,(SYSTEM+1) ┐
        PUSH    HL            ├ スタックの設定
        POP     SP            ┘
        DEC     H             ┐
        DEC     H             ├ (BUFEND)=SP-512
        LD      (BUFEND),HL   ┘
        LD      A,(DTA)       ┐
        OR      A             ├ コマンドの引数がなければUSAGEへ
        JP      Z,USAGE       ┘
        LD      A,WIDTH
        LD      HL,WIDTH * LINES
        LD      DE,FCB1       ┐
        CALL    FOPENR        │ ファイルをオープン．オープン
        OR      A             │ できなければNOFILEへ
        JP      NZ,NOFILE     ┘
        CALL    SETBUF        ┐
        OR      A             │ バッファの設定．設定でき
        JP      NZ,FILBIG     ┘ なければFILBIGへ
        CALL    INIT……………画面の初期化
        LD      HL,BUFTOP……バッファの先頭を表示
        LD      (TXTDSP),HL
        CALL    PUTBUF
LOOP:
        CALL    GETCOM…………コマンドを知る
        CP      'Q'           ┐
        JP      Z,BOOT        ┘ "Q"ならば終了
        CALL    COMEXEC………それ以外なら，コマンドの実行
        JR      LOOP……………繰り返し

;--- Abnormal End ---

USAGE:  ERREXT  M_USG……………引数がない場合
M_USG:  DB      CR,LF,"usage : less <filename>",CR,LF,EOS

NOFILE: ERREXT  M_NOFL…………ファイルが見つからない場合
M_NOFL: DB      CR,LF,"less : cannot open file.",CR,LF,EOS

FILBIG: ERREXT  M_FBIG…………ファイルが読み込めない場合
M_FBIG: DB      CR,LF,"less : file is too big.",CR,LF,EOS

;--- Command Exec ---……………コマンド実行ルーチン
;[IN]   A = COMMAND
COMEXEC:
        CP      'H'
        JR      NZ,COM_1
```

```
        LD      HL,BUFTOP
        LD      (TXTDSP),HL     } ファイルの先頭に移動(H)
        CALL    PUTBUF
        RET
COM_1:
        CP      'T'
        JR      NZ,COM_2
        LD      HL,(TXTDSP)
        LD      BC,WIDTH * LINES
        ADD     HL,BC
        LD      DE,(TXTEND)
        CPHLDE
        RET     NC              } ファイルの末尾に移動(T)
        LD      HL,(TXTEND)
        LD      BC,-WIDTH * LINES
        ADD     HL,BC
        LD      (TXTDSP),HL
        CALL    PUTBUF
        RET
COM_2:
        CP      CR
        JR      Z,COM_3
        CP      'J'
        JR      NZ,COM_4
COM_3:
        LD      HL,(TXTDSP)
        LD      BC,WIDTH * LINES
        ADD     HL,BC
        LD      DE,(TXTEND)
        CPHLDE
        RET     NC              } 1行進む(↵またはJ)
        LD      HL,(TXTDSP)
        LD      BC,WIDTH
        ADD     HL,BC
        LD      (TXTDSP),HL
        CALL    PUTBUF
        RET
COM_4:
        CP      'K'
        JR      NZ,COM_5
        LD      DE,(TXTDSP)
        LD      HL,BUFTOP
        CPHLDE
        RET     NC
        LD      HL,(TXTDSP)     } 1行戻る(K)
        LD      BC,-WIDTH
        ADD     HL,BC
        LD      (TXTDSP),HL
        CALL    PUTBUF
        RET
```

```
COM_5:
        CP      ' '
        JR      Z,COM_6
        CP      'F'
        JR      NZ,COM_8
COM_6:
        LD      HL,(TXTDSP)
        LD      BC,WIDTH * LINES
        ADD     HL,BC
        LD      DE,(TXTEND)
        CPHLDE
        RET     NC
        EX      DE,HL
        LD      HL,(TXTEND)
        LD      BC,-WIDTH * LINES           1ページ進む(SPACE または F)
        ADD     HL,BC
        CPHLDE
        JR      C,COM_7
        EX      DE,HL
COM_7:
        LD      (TXTDSP),HL
        CALL    PUTBUF
        RET
COM_8:
        CP      'B'
        RET     NZ
        LD      HL,(TXTDSP)
        LD      DE,BUFTOP
        OR      A
        SBC     HL,DE
        LD      BC,-WIDTH * LINES
        ADD     HL,BC
        EX      DE,HL                       1ページ戻る(B)
        JR      NC,COM_9
        LD      HL,(TXTDSP)
        ADD     HL,BC
COM_9:
        LD      (TXTDSP),HL
        CALL    PUTBUF
        RET

;--- Get Key --- ……………コマンドの文字を入力
;[OUT]  A = COMMAND
GETCOM:
        PUTMSG  M_PROM ………プロンプトの表示
        SYSCAL  Ø1H ……………1文字入力
```

```
        CP      'z'+1   ┐
        RET     NC      │
        CP      'a'     ├ 小文字を大文字に変換
        RET     C       │
        SUB     'a'-'A' ┘
        RET
M_PROM:……………プロンプトの文字列
        DB      ESC,'Y',LINES+20H,20H
        DB      '[HBKJFTQ] ? ',EOS

        END     MAIN
```

リスト 7.4 LESS.MAC

```
; BUF.MAC : Text Buffer Handler……………バッファを扱うモジュール

        EXTRN   WIDTH, LINES, VRAMAD……外部で定義されたシンボル
        EXTRN   LDIRVM, FGETC…………………外部で定義されたサブルーチン

        PUBLIC  BUFTOP…………………………………外部から参照されるラベル
        PUBLIC  TXTDSP, TXTEND, BUFEND………外部から参照されるワークエリア
        PUBLIC  SETBUF, PUTBUF……………………外部から呼ばれるサブルーチン

        .Z80

TAB     EQU     09H ┐
CR      EQU     0DH │
LF      EQU     0AH ├ 文字コードの定義
EOS     EQU     '$' │
ESC     EQU     1BH ┘

        DSEG

TXTDSP: DS      2 ……………表示位置のポインタ
TXTEND: DS      2 ……………テキストの最後
BUFEND: DS      2 ……………バッファの最後

        CSEG

;--- Read File and Set Buffer ---……………ファイルをバッファに読み込み，画面のイメージに展開
;[IN]   DE = FCB (Opend)
;[OUT]  A = 0 : (success) / 1 : (file big)
```

```
SETBUF:
        LD      HL,BUFTOP ··············HL＝読み込みポインタ
        LD      (TXTDSP),HL ···········表示位置ポインタの初期化
SB_LP1:
        LD      C,WIDTH ··············C＝行の残りの文字数
SB_LP2:                               （1行の文字数－X座標）
        PUSH    HL         ┐
        PUSH    BC         │
        LD      BC,(BUFEND)│
        OR      A          │読み込みポインタがバッファの
        SBC     HL,BC      │最後より大きければSB_BIGへ
        POP     BC         │
        POP     HL         │
        JR      NC,SB_BIG  ┘
        PUSH    HL
        PUSH    BC
        CALL    FGETC ··············1文字ファイルから読み込む
        POP     BC
        POP     HL
        JR      C,SB_EXIT ········ファイルの終わりならSB_EXITへ
        CP      7FH        ┐
        JR      Z,SB_LP2   │特殊な文字は読み飛ばす
        CP      ØFFH       │
        JR      Z,SB_LP2   ┘
        CP      ' '        ┐コントロールコード(00H～1FH)
        JR      C,SB_CTL   ┘ならSB_CTLへ
        LD      (HL),A     ┐
        INC     HL         │それ以外の文字はバッファに書き
        DEC     C          │込む．画面上の行末ならばSB_LP1
        JR      Z,SB_LP1   │へ、そうでなければSB_LP2へ
        JR      SB_LP2     ┘
SB_CTL: ···············コントロールコードの処理
        CP      LF
        JR      NZ,SB_TAB
        CALL    CRSUB      ┐改行(0AH)
        JR      SB_LP1     ┘
SB_TAB:
        CP      TAB
        JR      NZ,SB_LP2 ··············タブコードでなければ何もしない
        LD      A,WIDTH    ┐
        SUB     C          │E＝X座標
        LD      E,A        ┘
        AND     ØF8H       ┐A＝次のタブストップ(タブ
        ADD     A,8        ┘は8文字単位とする)
        SUB     E
```

```
SB_LP3:
        LD      (HL),' '   ┐
        INC     HL         │
        DEC     C          │次のタブストップまで
        JR      Z,SB_LP1   │を空白で満たす
        DEC     A          │
        JR      Z,SB_LP2   ┘
        JR      SB_LP3
SB_EXIT:
        CALL    CRSUB ……………ファイルの最後の行を空白で満たす
        LD      (TXTEND),HL ……テキストの終わりを設定
        XOR     A
        RET
SB_BIG: ……………ファイルが大きすぎてバッファに読み込めない場合
        LD      A,1
        RET
CRSUB: ……………HLの示すアドレスからCレジスタの示すバイト数分だけ空白で埋める
        LD      (HL),' '
        INC     HL
        DEC     C
        JR      NZ,CRSUB
        RET

PUTBUF: ……………バッファの内容を表示(バッファの構造はVRAMと同じなので,
        LD      HL,(TXTDSP)         そのままデータをVRAMに転送)
        LD      DE,VRAMAD
        LD      BC,WIDTH * LINES
        CALL    LDIRVM ……………VRAMにデータを転送
        RET
BUFTOP:
        DS      184Ø,' '
        END
```

リスト 7.5 BUF.MAC

```
; SCR.MAC : Direct Screen handler ……………画面を直接操作するためのモジュール

        EXTRN   WIDTH, LINES ……………外部で定義されたシンボル

        PUBLIC  VRAMAD ………………………外部で参照されるシンボル
        PUBLIC  INIT, LDIRVM ……………外部から呼ばれるサブルーチン

        .Z8Ø
```

```
RDSLT   EQU     ØØØCH
CALSLT  EQU     ØØ1CH
EXPTBL  EQU     ØFCC1H ……………BIOSのスロット番号が保存されている
VRAMAD  EQU     ØØØØH ……………テキスト画面のVRAM先頭アドレス

        CSEG

;--- Initilize Screen --- ……………画面の初期化
INIT:
        LD      A,WIDTH
        LD      (ØF3AEH),A ……………画面の横幅をワークエリアにセット
        XOR     A
        LD      IX,ØØ5FH ……………IX＝呼び出し先(画面モード切り換えルーチン)
        LD      IY,(EXPTBL-1)……IYレジスタの上位8ビットにEXPTBLの内容を代入
        CALL    CALSLT ……………BIOSをインタースロットコール
        LD      HL,VRAMAD
        LD      BC,WIDTH * LINES
        LD      A,' '
        LD      IX,ØØ56H ……………VRAMを同じデータで埋めるBIOS
        LD      IY,(EXPTBL-1)
        CALL    CALSLT ……………BIOSを呼び出す
        LD      A,(EXPTBL)
        LD      HL,ØØØ7H
        CALL    RDSLT ……………BIOSのROMの7番地にはVDPに直接書き込むときの
        LD      (WTPORT),A ……  I/Oポート番号がはいっている
        RET               ……VDPポート番号を保存

        DSEG

WTPORT: DS      1 ……………VDPに書き込むときのポート番号
                            (INITで設定される)

        CSEG

;--- Data Transfer Memory to VRAM --- ……………データをメモ
;[IN]   HL = Sour(MEM) , DE = Dest(VRAM)        りからVRAM
;       BC = length                             に転送する
LDIRVM:
        PUSH    HL
        PUSH    BC
        EX      DE,HL ……………HL＝VRAM(転送先)アドレス
        LD      IX,ØØ53H……………VDPをHLから書き込むよう設
        LD      IY,(EXPTBL-1) 定するBIOS
        CALL    CALSLT ……BIOSを呼び出す
        POP     DE ……………DE＝転送バイト数
        POP     HL ……………HL＝メモリ(転送元)アドレス
        LD      A,(WTPORT)
        LD      C,A ……………C＝VDP書き込みポート
        DI ……………割り込み禁止
```

```
LM1:
        OUTI
        DEC     DE          ┐
        LD      A,D         │ ポートCにメモリHLからDEバイト分出力
        OR      E           │
        JR      NZ,LM1      ┘
        EI ·············割り込み許可
        RET

        END
```

リスト 7.6 SCR.MAC

このプログラムのアセンブル／リンク・ロードを行うのですが，図 7.16 のように作成してください．

```
A>M8Ø =B:LESS ⏎

No Fatal error(s)

A>M8Ø =B:SCR ⏎

No Fatal error(s)

A>M8Ø =B:FILE ⏎

No Fatal error(s)

A>M8Ø =B:BUF ⏎

No Fatal error(s)                   ------BUFを一番最後にリンクしている

A>L8Ø B:LESS,B:SCR,B:FILE,B:BUF,B:LESS/N/E ⏎

MSX.L-8Ø  1.ØØ  Ø1-Apr-85  (c) 1981,1985 Microsoft

Data    Ø1Ø3    ØFB3    < 376Ø>

3961Ø Bytes Free
[Ø1Ø3   ØFB3        15]

A>█
```

図 7.16 LESS.COM の作成

このリンク・ロードの際に，BUF.RELを最後に読み込まなくてはならないことに注意してください．それは，プログラムの実行時にBUFモジュールの最後にあるラベルBUFTOP以降に読み込んだファイルの内容が置かれるからです．

では，実行のようすを見てみましょう（図7.17）．

```
        EXTRN   TXTDSP, TXTEND, BUFEND
        EXTRN   INIT, FOPENR
        EXTRN   SETBUF, PUTBUF

        PUBLIC  WIDTH, LINES

        .Z8Ø

WIDTH   EQU     8Ø
LINES   EQU     23

BOOT    EQU     ØØØØH
SYSTEM  EQU     ØØØ5H
FCB1    EQU     ØØ5CH
DTA     EQU     ØØ8ØH
CR      EQU     ØDH
LF      EQU     ØAH
EOS     EQU     '$'
ESC     EQU     1BH

[HBKJFTQ] ?■ ……………LESSのコマンド入力待ち
```

図7.17　LESS.COMの実行

MSX-DOSからでも直接ハードウェアを扱えることがわかってもらえたと思います．MSX-DOSからBIOSを呼び出すプログラムの作成は大変な作業ですが，MSXの機能を十分に活かしたプログラムを組むことは非常にやりがいがあることですし，その苦労の結果はかならず報われるはずです．

Appendix

BIOSエントリー一覧

ここに掲載したエントリーは，MSXのBIOSルーチンとして公開されているもののうち，ユーザープログラムの開発に有用と思われるルーチンを抜粋したものです．このBIOSを呼び出すための手順については本文の7章を参考にしてください．このエントリーでは以下に示すような表記を行います．

・機能　機能の解説
入力　呼び出すときに与えるパラメータ
出力　返されるパラメータ
使用　BIOSの内部で使用され，破壊されるレジスタ

●I／O初期設定

アドレス	ラベル名	機　　能
003EH	INIFNK	・ファンクションキーの内容を初期化 入力　なし 出力　なし 使用　すべて

●VDPへのアクセス

アドレス	ラベル名	機　　能
0041H	DISSCR	・画面表示の停止 入力　なし 出力　なし 使用　AF，BC
0044H	ENASCR	・画面の表示 入力　なし 出力　なし 使用　AF，BC
0047H	WRTVDP	・VDPのレジスタにデータを書き込む 入力　Bにデータ，Cにレジスタ番号 出力　なし 使用　AF，BC
004AH	RDVRM	・指定したアドレスのVRAMの内容を読む 入力　HL(アドレス) 出力　Aに読み出した値 使用　AF
004DH	WRTVRM	・指定したアドレスのVRAMにデータを書き込む 入力　HL(アドレス)，A(データ) 出力　なし 使用　AF
0050H	SETRD	・VDPにVRAMのアドレスを設定し読み込み状態にする 入力　HL(アドレス) 出力　なし 使用　AF
0053H	SETWRT	・VDPにVRAMのアドレスを設定し書き込み状態にする 入力　HL(アドレス) 出力　なし 使用　AF
0056H	FILVRM	・VRAMを同一データで埋める 入力　HL(先頭アドレス)，BC(長さ)，A(データ) 出力　なし 使用　AF，BC
0059H	LDIRMV	・VRAMからメモリにブロック転送 入力　HL(VRAMアドレス)，DE(転送先アドレス)，BC(長さ) 出力　なし 使用　すべて

アドレス	ラベル名	機　　能
005CH	LDIRVM	・メモリからVRAMにブロック転送 入力　HL(転送元アドレス), DE(VRAMアドレス), BC(長さ) 出力　なし 使用　すべて
005FH	CHGMOD	・画面モードの変更 入力　Aに画面モード 出力　なし 使用　すべて
0062H	CHGCLR	・画面の色を変える 入力　FORCLR(F3E9H)に前景色 BAKCLR(F3EAH)に背景色 BDRCLR(F3EBH)に周辺色 出力　なし 使用　すべて
0069H	CLRSPR	・すべてのスプライトを初期化 入力　SCRMOD(FCAFH)に画面モード 出力　なし 使用　すべて
0084H	CALPAT	・スプライト・ジェネレータ・テーブルのアドレスを求める 入力　Aにスプライトパターン番号 出力　HLにアドレス 使用　AF, DE, HL
0087H	CALATR	・スプライト・アトリビュート・テーブルのアドレスを求める 入力　Aにスプライト面番号 出力　HLにアドレス 使用　AF, DE, HL
008AH	GSPSIZ	・現在のスプライトの大きさを調べる 入力　なし 出力　Aにスプライトパターンの大きさ(バイト数), 16×16のときはCYフラグをセット 使用　AF
008DH	GRPPRT	・グラフィック画面に文字を表示 入力　Aに文字コード 出力　なし 使用　なし

●PSGへのアクセス

アドレス	ラベル名	機　　能
0090H	GICINI	・PSGを初期化しPLAY文のための初期値を設定 入力　なし 出力　なし 使用　すべて
0093H	WRTPSG	・PSGのレジスタにデータを書き込む 入力　AにPSGのレジスタ番号 Eにデータ 出力　なし 使用　なし
0096H	RDPSG	・PSGのレジスタの値を読む 入力　AにPSGのレジスタ番号 出力　Aに読み込んだ値 使用　なし

●キーボード，CRT，プリンタへの入出力

アドレス	ラベル名	機　　能
009CH	CHSNS	・キーボード・バッファの状態を調べる 入力　なし 出力　バッファが空ならZフラグをセット，それ以外ならZフラグをリセット 使用　AF
009FH	CHGET	・1文字入力(入力待ちあり) 入力　なし 出力　Aに文字コード 使用　AF
00A2H	CHPUT	・1文字表示 入力　Aに文字コード 出力　なし 使用　なし
00A5H	LPTOUT	・1文字プリンタ出力 入力　Aに文字コード 出力　失敗した場合はCYフラグ=1 使用　F
00A8H	LPTSTT	・プリンタの状態を調べる 入力　なし 出力　レディ状態ならA=255でZフラグをリセット，それ以外ならA=0でZフラグをセット 使用　AF
00ABH	CNVCHR	・グラフィック・ヘッダかどうかを調べてコードを変換 入力　Aに文字コード 出力　グラフィック・ヘッダならCYフラグのみリセット，変換されたグラフィック文字ならCYフラグとZフラグをセット，変換されていなければCYフラグをセット Zフラグをリセット 使用　AF
00B1H	BREAKX	・CTRL+STOPを押しているかを調べる 入力　なし 出力　押されていればCYフラグをセット 使用　AF
00C0H	BEEP	・ブザーを鳴らす 入力　なし 出力　なし 使用　すべて

アドレス	ラベル名	機　能
00C3H	CLS	・画面クリア 入力　なし 出力　なし 使用　AF, BC, DE
00C6H	POSIT	・カーソルの移動 入力　HにX座標, LにY座標 出力　なし 使用　AF
00CCH	ERAFNK	・ファンクションキーの表示を消す 入力　なし 出力　なし 使用　すべて
00CFH	DSPFNK	・ファンクションキーを表示 入力　なし 出力　なし 使用　すべて
00D2H	TOTEXT	・画面をテキストモードにする 入力　なし 出力　なし 使用　すべて

●汎用入出力ポートへのアクセス

アドレス	ラベル名	機　能
00D5H	GTSTCK	・ジョイスティックの状態を調べる 入力　Aに調べるジョイスティック番号(BASICのSTICK関数を参照) 出力　Aに方向(BASICのSTICK関数を参照) 使用　すべて
00D8H	GTTRIG	・トリガボタンの状態を調べる 入力　トリガボタン番号(BASICのSTRIG関数を参照) 出力　押されていればA=255, 押されてなければA=0 使用　AF

アドレス	ラベル名	機　能
00DBH	GTPAD	・タッチパッドの状態を調べる 入力　Aにタッチパッドの番号(BASICのPAD関数を参照) 出力　Aに値 使用　すべて
00DEH	GTPDL	・パドルの値を調べる 入力　Aにパドルの番号(BASICのPDL関数を参照) 出力　Aに値 使用　すべて

●その他のルーチン

アドレス	ラベル名	機　能
0132H	CHGCAP	・CAPSランプの状態を変える 入力　A=0ならランプを消す A=0ならランプを付ける 出力　なし 使用　AF
0135H	CHGSND	・1ビット・サウンドポートの状態を変える 入力　A=0ならOFF, A=0ならON 出力　なし 使用　AF
013EH	RDVDP	・VDPのステータスレジスタ(#0)の値を読む 入力　なし 出力　Aに読み込んだ値 使用　A
0141H	SNSMAT	・キーボード・マトリクスから指定した行の値を読む 入力　Aに指定する行 出力　Aにデータ(押されているキーに対応するビットが0) 使用　AF, C
0156H	KILBUF	・キーボード・バッファのクリア 入力　なし 出力　なし 使用　HL

MSX・M-80の代表的な擬似命令

●文の制御

擬似命令	機　能
ORG <式>	ロケーションカウンタの値を変更
END [<式>]	プログラムの終了
.Z80	Z80モードにする
.8080	8080モードにする
.PHASE <式>	実行時リロケート範囲の始まり
.DEPHASE	実行時リロケート範囲の終わり
.RADIX <値>	基数の指定

●データ定義／シンボル定義(詳しくは本文参照)

擬似命令	機　能
DB <データ列>	バイト単位のデータ定義
DW <データ列>	2バイト単位のデータ定義
DS <大きさ>	領域確保
DS <大きさ>, <値>	領域確保と初期値設定
<シンボル> ASET <値>	再定義可のシンボル定義
<シンボル> SET <値>	再定義可のシンボル定義
<シンボル> EQU <値>	再定義不可シンボル定義
PUBLIC <シンボル列>	外部シンボル宣言
EXTRN <シンボル列>	外部参照宣言

●ロケーション・モード関連(詳しくは本文参照)

擬似命令	機　　能
ASEG	絶対モード
CSEG	コード相対モード
DSEG	データ相対モード

●ファイル操作

擬似命令	機　　能
INCLUDE<ファイル名>	他のファイルを挿入

●マクロ関連

擬似命令	機　　能
MACRO<パラメータ列>	マクロ定義の開始
ENDM	マクロ定義の終了
EXITM	マクロの展開の終了
LOCAL<パラメータ列>	仮パラメータを独自なものへ置換するように指示
REPT<反復回数>	反復ブロックの開始

MSX-DOSシステムコール一覧

・設　定　システムコールを行う前にレジスタやメモリにセットする値
・戻り値　システムコールから戻ったときにセットされている値
＊1 本書で使用したもの
＊2 CP／Mとの互換性がないもの

No.	機　　能
00H	システムリセット 設　定　なし 戻り値　なし 備　考　MSX-DOS上からコールするとシステムがリセットされる。MSX DISK-BASICからコールするとDISK-BASICがウォームスタート
01H ＊1	コンソール入力 設　定　なし 戻り値　A←コンソールから入力した1文字 備　考　入力なき場合入力待ち，入力文字はコンソールにエコーバックCTRL-C,CTRL-P,CTRL-Nの処理を行う
02H ＊1	コンソール出力 設　定　E←出力する文字コード 戻り値　なし 備　考　CTRL-Sの入力があった場合はその処理を行う
03H	外部入力 設　定　なし 戻り値　A←AUXデバイスから読み込んだ1文字
04H	外部出力 設　定　E←AUXデバイスに出力する1文字 戻り値　なし
05H	プリンタ出力 設　定　E←プリンタに出力する文字コード 戻り値　なし
06H	直接コンソール入力出 設　定　E←FFHコンソール入力 E←FFH以外，その値を文字コードとみなしコンソール出力 戻り値　入力の場合　A←入力した1文字(入力なき場合　A←00H) 出力の場合　なし 備　考　コントロール文字の処理，エコーバックは行わない
07H	直接コントロール入力　その1 設　定　なし 戻り値　A←コンソールから入力した1文字 備　考　コントロール文字の処理，エコーバックは行わない
08H	直接コンソール入力　その2 設　定　なし 戻り値　A←コンソールから入力した1文字 備　考　コントロール文字の処理はNo.01Hと同様に行い，エコーバックは行わない
09H ＊1	文字列出力 設　定　DE←メモリ上に用意した，コンソールに出力するべき文字列の先頭アドレス 戻り値　なし 備　考　文字列の最後にEOFを付ける必要ありCTRL-Sを処理する
0AH	文字列入力 設　定　DE←最大入力文字数(1～FFH)　をセットしたメモリのアドレス 戻り値　DEにセットしたアドレス+1←実際に入力された文字数，DEにセットしたアドレス+2以降←コンソールから入力された文字列 備　考　入力文字数が指定した最大に満たない場合，リターンコードを入力の終端とみなす．入力時にテンプレートによる編集可

No.	機　能
0BH ＊1	**コンソールの状態チェック** **設　定**　なし **戻り値**　キーが押されている場合A←FFH 押されていない場合A←00H
0CH ＊2	**バージョン番号の獲得** **設　定**　なし **戻り値**　HL←0022H **備　考**　CP／M用，MSX-DOSでは一律に0022Hがセットされる
0DH	**ディスクリセット** **設　定**　なし **戻り値**　なし **備　考**　変更されたがまだディスクに書き込まれていないセクタがあれば，これを書き込み，デフォルト・ドライブをドライブAに設定，DTAを0080Hにセット
0EH	**デフォルト・ドライブの設定** **設　定**　E←デフォルト・ドライブ番号 (A=00H, B=01H, …, H=07H) **戻り値**　なし
0FH ＊1	**ファイルのオープン** **設　定**　DE←オープンされていないファイルのFCBの先頭アドレス **戻り値**　成功した場合A←00H，指定のFCB←各フィールドの設定／失敗した場合A←FFH
10H ＊1	**ファイルのクローズ** **設　定**　DE←オープンされたファイルのFCBの先頭アドレス **戻り値**　成功した場合A←00H／失敗した場合A←FFH
11H ＊1	**ファイルの検索　その1** **設　定**　DE←オープンされていないファイルのFCBの先頭アドレス **戻り値**　見つかった場合A←00H，DTAで示される領域←そのファイルのディスク上のドライブ番号とディレクトリ・エントリ，FCBにドライブ番号をセット／見つからなかった場合A←FFH
12H ＊1	**ファイルの検索　その2** **設　定**　なし **戻り値**　見つかった場合A←00H，DTAで示される領域←そのファイルのドライブ番号とディレクトリ・エントリ／見つからなかった場合A←FFH
13H	**ファイルの抹消** **設　定**　DE←オープンされたファイルのFCBの先頭アドレス **戻り値**　成功した場合A←00H 失敗した場合A←FFH
14H ＊1	**シーケンシャルな読み出し** **設　定**　DE←オープンされたファイルのFCBアドレス，FCBのカレントブロック←読み出し開始ブロック，FCBのカレントレコード←読み出し開始レコード **戻り値**　成功した場合A←00H，DTAで示される領域←読み込んだ1レコード／失敗した場合A←01H
15H	**シーケンシャルな書き込み** **設　定**　DE←オープンされたファイルのFCBの先頭アドレス，FCBのカレントブロック←書き込み開始ブロック，FCBのカレントレコード←書き込み開始レコード，DTA以降の128バイト←書き込むデータ **戻り値**　成功した場合A←00H／失敗した場合A←FFH
16H ＊1	**ファイルの作成** **設　定**　DE←オープンされていないファイルのFCBの先頭アドレス **戻り値**　成功した場合A←00H 失敗した場合A←FFH
17H ＊1	**ファイル名の変更** **設　定**　変更したいファイルのFCB←新ファイル名，DE←そのFCBの先頭アドレス **戻り値**　成功した場合A←00H 失敗した場合A←FFH
18H	**ログイン・ベクトルの獲得** **設　定**　なし **戻り値**　HL←オンライン・ドライブ情報 **備　考**　戻り値のうち，Lレジスタの各ビットがドライブ番号に対応，1ならオンライン，0ならオフライン
18H	**デフォルト・ドライブ番号の獲得** **設　定**　なし **戻り値**　A←デフォルト・ドライブ番号 (A=00H, B=01H, …H=7H)
1AH ＊1	**DTAの設定** **設　定**　DE←転送先アドレス(DTAアドレス) **戻り値**　なし
1BH ＊2	**ディスク情報の獲得** **設　定**　E←目的のディスクがはいっているドライブ番号 **戻り値**　A←1クラスタあたりの論理セクタ数，BC←論理セクタのサイズ，DE←クラスタの総数，HL←未使用クラスタの総数，IX←DPBの先頭アドレス，IY←メモリ上のFATの先頭アドレス

No.	機　　能
21H	**ランダムな読み出し** **設　定**　DE←オープンされたファイルのFCBの先頭アドレス，FCBのランダムレコード←読み出すレコード **戻り値**　成功した場合A←00H，DTAで示される領域←読み込んだ1レコードの内容 **備　考**　レコードの大きさは128バイト固定長
22H	**ランダムな書き込み** **設　定**　DE←オープンされたファイルのFCBの先頭アドレス，FCBのランダムレコード←書き込むレコード，DTA以降の128バイト←書き込むデータ **戻り値**　成功した場合A←00H／失敗した場合A←01H **備　考**　レコードの大きさは128バイト固定長
23H	**ファイルサイズの獲得** **設　定**　DE←オープンされたファイルのFCBの先頭アドレス **戻り値**　成功した場合A←00H，FCBのランダムレコード・フィールド←指定されたファイルのサイズ／失敗した場合A←FFH **備　考**　ファイルのサイズは128バイト単位で計算するため，200バイトなら2，257バイトなら3をセット
24H	**ランダムレコード・フィールドの設定** **設　定**　DE←オープンされたファイルのFCBの先頭アドレス，FCBのカレントブロック←目的のブロック，FCBのカレントレコード←目的のレコード **戻り値**　ランダムレコード・フィールド←指定されたFCBのカレントブロック・フィールドおよびカレントレコード・フィールドから計算したカレントレコード・ポジション
26H ＊2	**ランダムな書き込み　その2** **(ランダム・ブロック・アクセス)** **設　定**　DE←オープンされたファイルのFCBの先頭アドレス，FCBのレコードサイズ←書き込むレコードサイズ，FCBのランダムレコード←書き込み開始レコード，HL←書き込むレコード数，DTA以降のメモリ領域←書き込むデータ **戻り値**　成功した場合A←00H／失敗した場合A←FFH
27H	**ランダムな読み出し　その2** **(ランダム・ブロック・アクセス)** **設　定**　DE←オープンされたファイルのFCBの先頭アドレス，FCBのレコードサイズ←読み出すレコードサイズ，FCBのランダムレコード←読み出し開始レコード，HL←読み出すレコード数 **戻り値**　成功した場合A←00H，HLに実際に読み込んだレコード数／ 失敗した場合A←FFH
28H	**ランダムな書き込み　その3** **設　定**　DE←オープンされたファイルのFCBの先頭アドレス，FCBのランダムレコード←書き込むレコード，DTA以降の128バイト←書き込むデータ **戻り値**　成功した場合A←00H／失敗した場合A←01H **備　考**　レコードの大きさは128バイト固定長，No.22Hと違い書き込むレコード番号がファイルの持つレコード数より大きい場合も使える
2AH ＊2	**日付の獲得** **設　定**　なし **戻り値**　HL←年，D←月，E←日，A←曜日
2BH ＊2	**日付の設定** **設　定**　HL←年，D←月，E←日 **戻り値**　成功した場合A←00H／失敗した場合A←FFH
2CH ＊1 ＊2	**時刻の獲得** **設　定**　なし **戻り値**　H←時，L←分，D←秒，E←1／100秒
2DH ＊2	**時刻の設定** **設　定**　H←時，L←分，D←秒，E←1／100秒 **戻り値**　成功した場合A←00H／失敗した場合A←FFH
2EH ＊2	**ベリファイ・フラグの設定** **設　定**　ベリファイ・フラグをセットする場合E←00H／リセットする場合E←FFH **戻り値**　なし
2FH ＊2	**論理セクタを用いた読み出し** **設　定**　DE←読み出しを開始する論理セクタ番号，H←読み出す論理セクタの数，L←読み出すディスク・ドライブの番号 **戻り値**　DTAバッファ←読ゝ込んだ内容
30H ＊2	**論理セクタを用いた書き込み** **設　定**　DE←書き込みを開始する論理セクタ番号，H←書き込む論理セクタの数L←書き込むディスク・ドライブの番号，DTAで示されるアドレス以降のメモリ領域←書き込む内容 **戻り値**　なし

MEDの代表的なコマンド

●ダイレクトコマンド

（CTRL と文字キーを同時に押すもの）

コマンド	機能
CTRL ＋ K	カーソルを画面の左上隅に移動
CTRL ＋ T	カーソルを画面の左下隅に移動
CTRL ＋ F	カーソルを１単語分，右に移動
CTRL ＋ B	カーソルを１単語分，左に移動
CTRL ＋ C	カーソルを行の先頭に移動
CTRL ＋ V	カーソルを行の末尾に移動
CTRL ＋ U	画面を１行分，上にスクロール
CTRL ＋ D	画面を１行分，下にスクロール
CTRL ＋ W	画面を20行分，上にスクロール
CTRL ＋ Z	画面を20行分，下にスクロール
CTRL ＋ Y	カーソルの行を削除
CTRL ＋ E	カーソル位置から行末まで削除
CTRL ＋ O	カーソル位置に空行を挿入
CTRL ＋ P	ヤンクバッファの内容を挿入

●ファンクションキー

キー	機能
F1	テキストの最初の行にカーソルを移動
F2	テキストの最後の行にカーソルを移動
F3	文字列の検索
F4	文字列の置換
F5	最後におこなった検索／置換を再実行
F6	コマンドの一覧を表示

●インダイレクトコマンド

ESC を押したのち，〈コマンド名〉＋⏎で実行される

〈コマンド名〉	機能
0	最初の行にカーソルを移動
$	最後の行にカーソルを移動
行番号	指定行にカーソルを移動
回数 ＋ F3 ＋ ↓	上方向に文字列の検索
回数 ＋ F3 ＋ ↑	下方向に文字列の検索
回数 ＋ F3 ＋ ⏎	前回と同じ方向に検索
回数 ＋ F4	文字列の置換
回数 ＋ F5	検索／置換を再実行
READ	テキストを読み込む
DIR	ディレクトリの表示
HELP	コマンド一覧の表示
NEW	エディットバッファ消す
QUIT	エディタの終了
UPDATE	ファイルの更新

●ブロックコマンド

SELECT を押し，ブロックの始点／終点を指定したのち，コマンド文字を入力

コマンド文字	機能
C	他の場所に複写
D	ヤンクバッファに移動
B	ヤンクバッファに複写
W	ファイルに書き込む
A	ファイルに追加

MSX・M-80／MSX・L-80スイッチ一覧

●MSX・M-80のスイッチ

スイッチ	機　　能
／O	リスティング・ファイルのアドレスを8進数表示
／R	リロケータブル・ファイルを作成
／L	リスティング・ファイルを作成
／C	クロスリファレンス・ファイルを作成
／Z	Z80オペコードのアセンブル
／I	8080オペコードのアセンブル
／M	DS擬似命令の領域を0に初期設定

●MSX・L-80のスイッチ

スイッチ	機　　能
／G	リンク・ロード後プログラムを実行
／G：〈名前〉	リンク・ロード後プログラムを指定した外部シンボルから実行
／E	リンク・ロード後，OSに戻る
／E：〈名前〉	指定した外部シンボルを実行アドレスに設定し，OSに戻る
／N	／Nの直前のファイル名でプログラムをセーブ
／N：P	プログラム領域だけをセーブ
／S	／Sの直前のファイル名からオブジェクト・ライブラリを検索
／U	未定義の外部参照名をリスティング
／M	外部参照マップをリスティング
／O	基数を8進にする
／H	基数を16進にする
／P	プログラムの開始アドレス設定
／D	データ領域の開始アドレス設定
／R	MSX・L-80を初期状態に戻す
／X	インテルHEXファイルを作成
／Y	シンボル・ファイルを出力

索　引

ナ

ハ

マ

ヤ

ラ

ワ

参考文献

「はじめて読むアセンブラ」　村瀬康治著
「実習C言語」　三田典玄著
「MSXマシン語入門講座」　湯浅敬著
「MSX2 BASIC入門」　アスキー書籍編集部編著
「MSX2テクニカルハンドブック」　アスキー監修
「MSX-DOS入門」　中村哲著
「MSX-DOSスーパーハンドブック」　BITS著　MSXマガジン監修
「MSX-DOS TOOLS USR'S MANUAL」
「MSX-S BUG USER'S MANUAL」　以上（株）アスキー
「ソフトウェア作法」　B. W.カーニハン／P.J.プローガー著　木村泉訳
共立出版

MSX-DOSアセンブラプログラミング

1988年6月11日　初版発行
1989年6月1日　第1版第2刷発行
定価1,240円（本体1,204円）

著　者　蔭山哲也（かげやまてつや）
発行者　塚本慶一郎
発行所　株式会社アスキー
〒107-24　東京都港区南青山6-11-1スリーエフ南青山ビル
振　替　東京4-161144
TEL (03)486-7111（大代表）
情報TEL (03)498-0299（ダイヤルイン）
出版営業部TEL (03)486-1977（ダイヤルイン）

制　作　株式会社GARO
印　刷　モリモト印刷株式会社

編集　佐藤英一・竹内充彦
本文デザイン　川戸明子

ISBN4-87148-303-7 C3055 P1240E